NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NPSNET: FLIGHT SIMULATION DYNAMIC MODELING USING QUATERNIONS

by

Joseph M. Cooke

March 1992

Thesis Advisor: Dr. Michael Zyda
Co-Advisor: David R. Pratt

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS/CK	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
NPSNET: FLIGHT SIMULATION DYNAMIC MODELING USING QUATERNIONS(U)

12. PERSONAL AUTHOR(S)
Cooke, Joseph M.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	From 03/90 To 03/92	March 1992	62

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer Graphics, Animation, Simulation, Aerodynamics, Quaternions

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Naval Postgraduate School (NPS) has actively explored the design and implementation of networked, real-time, three-dimensional battlefield simulations on low cost, commercially available graphics workstations. The most recent system, NPSNET, has improved in functionality to such an extent, that it is considered a low cost version of the Defense Advanced Research Project Agency's (DARPA) SIMNET system. In order to reach that level, it was necessary to economize in certain areas of the code so that real time performance occurred at an acceptable level. One of those areas was in aircraft dynamics. However, with "off-the-shelf" computers becoming faster and cheaper, real-time and realistic dynamics are no longer an expensive option. The realistic behavior can now be enhanced through the incorporation of an aerodynamic model. To accomplish this task, a prototype flight simulator was built that is capable of simulating numerous types of aircraft simultaneously within a virtual world. Beside being easily incorporated into NPSNET, such a simulator will also provide the base functionality for the creation of a general purpose aerodynamic simulator that is particularly useful to aerodynamic students for graphically analyzing differing aircraft's stability and control characteristics. This system is designed for use on a Silicon Graphics workstation and uses the GL libraries.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Michael J. Zyda	22b. TELEPHONE (Include Area Code): (408) 646-2305; 22c. OFFICE SYMBOL: CS/ZK

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete

Approved for public release; distribution is unlimited

NPSNET: FLIGHT SIMULATION DYNAMIC MODELING USING QUATERNIONS

by

Joseph M. Cooke
Major, United States Marine Corps
B.S., Syracuse University, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

ABSTRACT

The Naval Postgraduate School (NPS) has actively explored the design and implementation of networked, real time, three-dimensional battlefield simulations on low cost, commercially available graphics workstations. The most recent system, NPSNET, has improved in functionality to such an extent, that it is considered a low cost version of the Defense Advanced Research Project Agency's (DARPA) SIMNET system. In order to reach that level, it was necessary to economize in certain areas of the code so that real time performance occurred at an acceptable level. One of those areas was in aircraft dynamics. However, with "off-the-shelf" computers becoming faster and cheaper, real-time and realistic dynamics are no longer an expensive option. The realistic behavior can now be enhanced through the incorporation of an aerodynamic model. To accomplish this task, a prototype flight simulator was built that is capable of simulating numerous types of aircraft simultaneously within a virtual world. Beside being easily incorporated into NPSNET, such a simulator will also provide the base functionality for the creation of a general purpose aerodynamic simulator that is particularly useful to aerodynamic students for graphically analyzing differing aircraft's stability and control characteristics. This system is designed for use on a Silicon Graphics workstation and uses the GL libraries.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. MOTIVATION

The current state of the art in simulation technology has provided today's military with many extremely valuable training experiences that could not have been obtained elsewhere and, as a result, has greatly increased survivability and readiness. From flight simulators, which allow a pilot to explore the edge of the flight envelope without endangering crew or multi-million dollar assets, to battlefield simulators, which allow entire fighting divisions to practice command and control without having to incur the enormous costs of running a full blown field exercise, computer simulation has become a way of doing business within the military.

One simulation system designed by the Defense Advanced Research Projects Agency (DARPA) is the Simulation Networking (SIMNET) [Thorpe,1987]. SIMNET is a networked battlefield simulator that allows multiple use interaction on the battlefield at many different levels. Vehicle simulators, such as tanks and aircraft, connect to the network and become part of a three dimensional world. On the lowest level, this system allows the vehicle drivers a chance to study defensive and offensive tactics. On higher levels, Company, Battalion and Regiment Commanders can experiment with the movement of forces, thereby learning what will and won't work at their particular level.

At the Naval Postgraduate School (NPS), an effort to develop a SIMNET type system based on commercially available, general purpose, graphic workstations, has been active for a number of years. This system, NPSNET, consists of Silicon Graphics workstations attached to a local area Ethernet [Zyda,et al.,1992]. Eventually, NPSNET will become a node on the SIMNET network.

Over the past few years, as workstations became available that were both faster and cheaper, capabilities, that were once too computationally expensive to incorporate, have become feasible. One such feature is realistic vehicle dynamics. As a result it is now possible to enhance the behavior of the system by the incorporation of realistic vehicle dynamics. A ground vehicle dynamics model currently in development at NPS will provide a greatly improved system for ground vehicle interaction with the enviornment. Because aircraft dynamics differs in many respects to ground vehicle dynamics, a separate dynamic and orientation model for air vehicles is required. This work provides such a model for incorporation into NPSNET.

Most of the research and development of aerodynamic simulators has involved either the development of large multi-million single aircraft platforms utilizing specialized computer hardware to increase processing speed, or the development highly proprietary flight simulation programs for use on a personal computers. As a result, a gap occurred where a general purpose flight simulation program operating on low cost graphics workstations, is needed. A system that allows the flight characteristics of any aircraft be modeled on a low cost, general purpose, graphical workstation, would be of great use to aerodynamic students in studying the stability and control of different aircraft.

B. FOCUS

There are several issues to be addressed when incorporating an aerodynamic model into a computer simulation. The complexity of the aerodynamic model, which orientation model to use and how aircraft data should be represented in the system are the critical issues and are center of focus in this work.

1. Complexity of the Aerodynamic Model

Of primary importance is that the complexity of the model fit the objective of the simulation. A complete aerodynamic model that includes fully articulated control surfaces

and airflow divergence patterns over the aircraft would seriously affect real time performance on any computer. Such models are usually computed in non-real time on super computers and are not appropriate for use on low cost graphics workstations. On the other hand, modeling the dynamics of an aircraft kinematically so that the aircraft's velocity and orientation are a linear result of control input, does not reproduce the nuances of aircraft motion and response that a user of a flight simulation would expect. The aerodynamic model's complexity must provide as much realism as possible without reducing the frame rate below an acceptable level. Therefore, the focus in creating the aerodynamic model is to create an aerodynamic model of sufficient complexity that provides as much aerodynamic realism as possible without adversely effecting real-time performance.

2. An Appropriate Orientation Model

The choice of rotational model is fundamentally limited to either an Euler angle or quaternion approach. This has been the subject of heated debate among computer scientists as to which rotation model is the most appropriate [Goldiez and Lin,1991]. Basically, either model can be used to direct orientation, but, depending on the type of vehicle being modeled, one method has certain advantages over the other. The primary focus in defining an appropriate orientation model is in determining which provides the right tool for the job [Shoemake,1985].

3. Defining Multiple Aircraft Aerodynamics Characteristics

A general purpose flight simulator, such as is to be incorporated into NPSNET, should be capable of simulating an unlimited number of different aircraft types. Because each type of aircraft exhibits its own specific aerodynamics and handling characteristics, it is desirable to change these characteristics depending on which type of aircraft is to be piloted. One solution is to base the aerodynamic model on the aircraft's stability coefficients, inertial coefficients, and airframe specifications, all of which are generally available by consulting aerodynamic stability and control textbooks. Stability coefficients

provide a very accurate model of aircraft flight behavior. However, care must be taken when modeling some of the newer generation fighters. To improve maneuverability, these aircraft have been designed aerodynamically unstable. Their stability coefficients reflect this instability.

C. SUMMARY OF CHAPTERS

Chapter II covers those considerations for incorporating an aerodynamic model into NPSNET. A detailed aerodynamic model utilizing stability coefficients is presented. Chapter III investigates the difference between Euler angle and quaternion representations for defining rotations. The advantages and disadvantages of both methods are discussed and a model is presented for integration into NPSNET. Chapter IV provides implementation details of the aerodynamic and orientation model into the prototype flight simulator. Chapter V covers conclusions and lists requirements and suggestions for future work in this area.

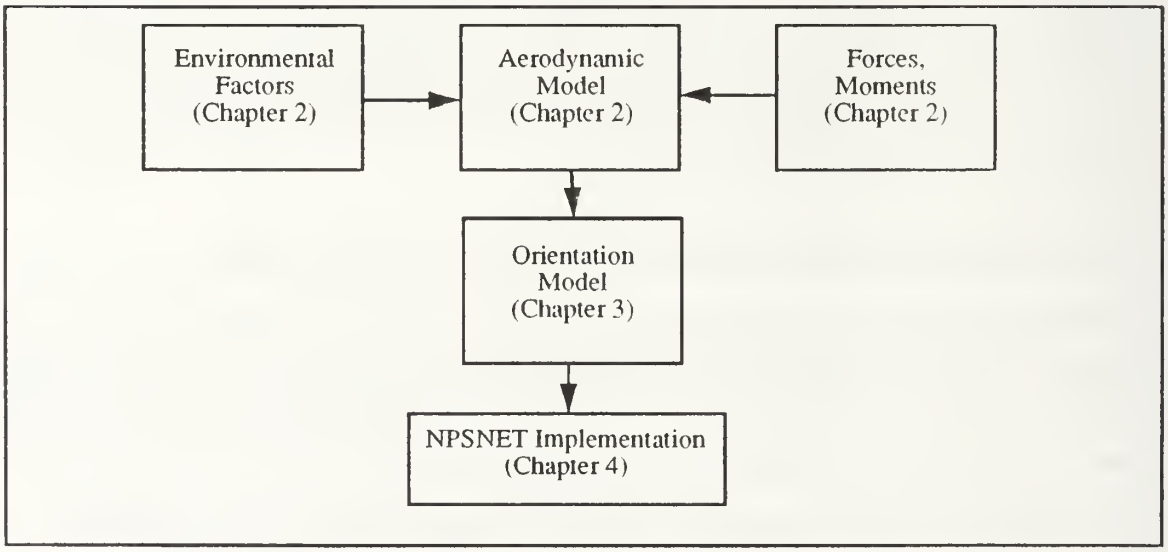

Figure 1.1: Flow of Data and Summary of Thesis Chapters

II. AERODYNAMIC MODEL

A. INTRODUCTION

In modeling the dynamic behavior of an aircraft within the framework of a computer based flight simulation, one usually begins by creating a mathematical model. This model represents the relationship between an aircraft and its interaction with the air mass in which it operates. Depending on the complexity of the model, additional forces, such as those associated with propulsion and environment, and complex systems, such as flight controls and weapons, are included [Rolfe,1986].

How detailed this model becomes is dependent on how it will be utilized. Generally, aerodynamic engineers will break the model down into components and study the results of detailed, but partial simulations. In complex flight simulators, such as those utilized to train pilots and aircrew, the model becomes more complex due to the increased number of systems that must be simulated and the response speed that is required for proper feedback to the pilot. As a result, the model loses fidelity, with the amount of detail limited by hardware and real-time constraints.

Flight dynamics in NPSNET requires a mathematical model similar to that found in complex flight simulators. The framework for such a model is presented in Figure 2.1. By adjusting the detail of those functions which surround the basic mathematical model, a simulation is built which can be made more or less detailed based on the speed of the workstation on which it is implemented.

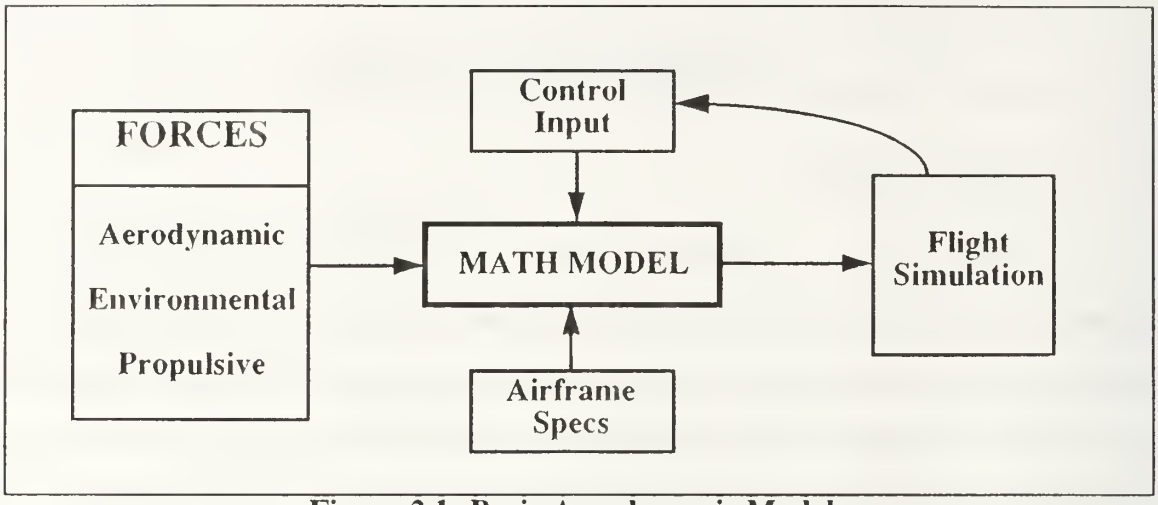

Figure 2.1: Basic Aerodynamic Model

B. COORDINATE SYSTEM

Coordinate systems and the method in which they are described vary to a great extent on the application and the preference of the user. In aircraft simulations, coordinate systems fall into two broad classes, "body" coordinates and "earth" or "inertial" coordinates [Rolfe,1986]. Body coordinates have their origin based at the center of gravity and continually move with the aircraft. Inertial coordinates, on the other hand, are defined with respect to the earth and have their origin positioned at some suitable location such as the center of the simulated world. Other coordinate systems exist, based on parameters such as the flight path and angle of attack. However, the aerodynamic model presented in this paper is based on geometric body and world coordinate systems. In general, all aerodynamic forces, accelerations and velocities are calculated in the body coordinate system first, and then converted to the world coordinate system prior to updating an aircraft's position and attitude.

Figure 2.2 shows the generally accepted convention for labeling of the axes in the two coordinate systems found in most aerodynamic textbooks [Anderson,1989]. Body coordinates are defined with the origin at the center of gravity (CG), the x axis along the

fuselage pointing out the nose of the aircraft, the y axis along the wing-line pointing out the right wing, and the z axis pointing out the bottom of the plane. World coordinates are defined with the origin based at a fixed point on the ground, the x axis pointing North, the y axis pointing East, and the z axis pointing down. Because of its limited effect, the curvature of the earth is ignored.

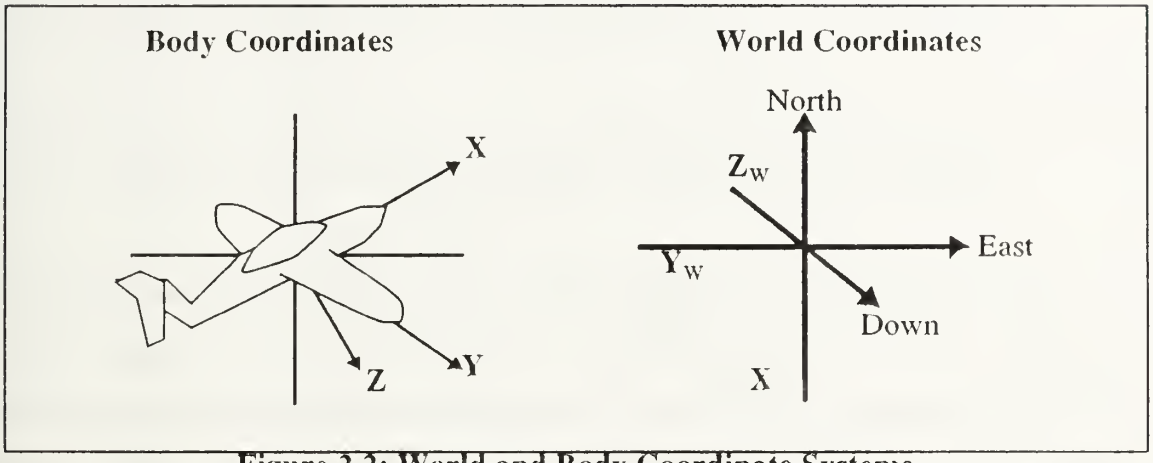

Figure 2.2: World and Body Coordinate Systems

C. DEFINITION OF TERMS

In a dynamics model, velocities, accelerations and forces are described in both world and body coordinate systems. Without an explicit description of the variables used to describe the model, confusion can arise. Most terms described in this paper refer to the

geometric body axes. However if a reference is made to the world coordinate system the subscript "w" is used. See Figure 2.3 for terms defined in world coordinates.

X_w, Y_w, Z_w	Aircraft location in world coordinates (feet)
U_w, V_w, W_w	Aircraft velocity in world coordinates (ft/sec)

Figure 2.3: Terms Defined within the World Coordinate System

Terms without the "w" subscript relate to body axes and include linear and angular velocities, accelerations, forces, moments, angle of attack and sideslip angle. Figure 2.4 provides a description of these terms. Note that the direction of angular accelerations and velocities and moment terms are defined using the right hand rule around their respective axis (Figure 2.5).

U, V, W	Linear velocity along X, Y, and Z body axes (ft/sec)
P, Q, R	Angular velocity along X,Y, and Z body axes (rad/sec)
V_τ	Resultant velocity Vector ($U^2 + V^2 + W^2$)
V_ε	Wind velocity across tail of aircraft
$\dot{U}$, $\dot{V}$, $\dot{W}$	Linear acceleration (ft/sec^2)
$\dot{P}$, $\dot{Q}$, $\dot{R}$	Angular acceleration (rad/sec^2)
F_x, F_y, F_z	Forces acting on aircraft
L, M, N	Moments about the X, Y, and Z axes
α	Angle of attack[$\tan^{-1}$ (W/U)]
β	Sideslip [$\tan^{-1}$ (V/U)]

Figure 2.4: Terms Defined within the Aircraft Body Coordinate System

The aircraft control surfaces such as elevator, ailerons, and rudder are defined as a rotation in radians around their respective hinge points on the aircraft. When a control surface is flush with the aircraft, the angle of deflection is zero. See Figure 2.6 for a further description.

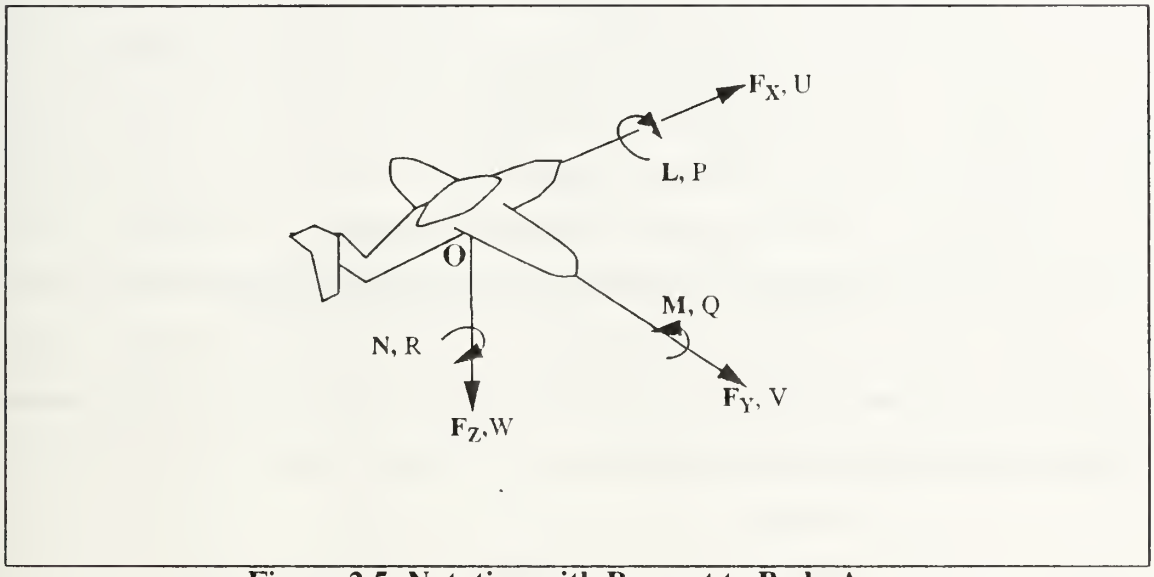

Figure 2.5: Notation with Respect to Body Axes

δe	elevator deflection positive down (radians) a positive δe produces a positive lift and a negative pitch moment
δa	aileron deflection positive left (radians) a positive δa produces a negative roll moment
δr	positive nose left a positive δr produces (radians) a positive sideforce and a negative yaw moment

Figure 2.6: Terminology Defining Aircraft Controls

Within the aerodynamic model, the particular aircraft being modeled is characterized by certain dimensional characteristics. A description of these terms is included in Figure 2.7.

S	surface area of wing (ft^2)
b	wing span (ft)
c	cord length (ft)
w	weight (lbs)
I_{xx}	roll inertia (slug-ft^2)
I_{yy}	pitch inertia (slug-ft^2)
I_{zz}	yaw inertia (slug-ft^2)
I_{xz}	roll-yaw cross inertia (slug-ft^2)

Figure 2.7: Aircraft Dimensional Specifications

The model presented in this paper utilizes English Standard Units of measurement. Therefore, the following units are utilized:

Distance:	feet
Force:	pounds
Mass:	slugs
Angles:	radians
Temperature:	Kelvin
Time:	seconds

In NPSNET distances are measured by meters. A conversion can be made into the metric system by either changing all the units of measurement to standard metric units prior to calculating any result, or converting the final result to meters prior to updating position using a simple conversion factor of 0.3048 meters/feet. In this paper the latter method is utilized.

D. MODEL DESCRIPTION

As indicated in Figure 2.1, the mathematical model presented takes forces, control inputs and aircraft specifications as inputs, generating linear and angular velocities in aircraft body coordinates as outputs. Based on a classical representation of linear aerodynamics and utilizing the total force Equations (2.18 - 2.20), all the forces associated with lift and drag are calculated utilizing aerodynamic stability derivatives [Roskam,1979]. Stability derivatives, first used by Bryan over a half-century ago, assume that all aerodynamic forces and moments can be expressed as a function of the instantaneous value of the perturbation variables [Nelson,1989]. The perturbation variables are the instantaneous changes from the reference conditions of translational velocities, angular velocities, control deflections, and their derivatives. For example, the term $\delta X/\delta u$ is the stability derivative defining the change in X force with respect to the change in forward speed. This derivative can be expressed in terms of a non-dimensional coefficient C_{Xu} as follows:

$$\frac{\delta X}{\delta u} = C_{Xu}\frac{1}{u_o}QS \tag{2.1}$$

where

$$C_{Xu} = \frac{\delta C_X}{\delta (u/u_o)} \tag{2.2}$$

and Q is the dynamic pressure $1/2\rho V_\tau^2$.

Figure 2.8 lists the non-dimensional coefficients used in this model. these coefficients are generally broken down into three categories, lateral, longitudinal and control. The longitudinal coefficients represent forces effecting the longitudinal axes of the aircraft, while the lateral coefficients represent forces affecting the lateral axes of the aircraft. Non-dimensional coefficients, generated in actual aircraft testing, are available for most aircraft. By using these coefficients in combination with the dynamics equations, it is possible to build a general use flight simulator.

Longitudinal Coefficients

C_{Lo}	Reference Lift at zero angle of attack
C_{Do}	Reference Drag at zero angle of attack
$C_{L\alpha}$	Lift curve slope
$C_{D\alpha}$	Drag curve slope
C_{Mo}	Pitch moment
$C_{M\alpha}$	Pitch moment due to angle of attack
C_{LQ}	Lift due to pitch rate
C_{MQ}	Pitch moment due to pitch rate
$C_{L\dot{\alpha}}$	Lift due to angle of attack rate
$C_{M\dot{\alpha}}$	Pitch moment due to angle of attack rate

Lateral Coefficients

$C_{Y\beta}$	Side force due to sideslip
$C_{L\beta}$	Dihedral effect
C_{LP}	Roll damping
C_{LR}	Roll due to yaw rate
$C_{N\beta}$	Weather cocking stability
C_{NP}	Rudder adverse yaw
C_{NR}	Yaw dampening

Control Coefficients

$C_{L\delta e}$	Lift due to elevator
$C_{D\delta e}$	Drag due to elevator
$C_{M\delta e}$	Pitch due to elevator
$C_{L\delta a}$	Roll due to aileron
$C_{L\delta r}$	Roll due to rudder
$C_{N\delta a}$	Yaw due to aileron
$C_{N\delta r}$	Yaw due to rudder

Figure 2.8: Aircraft Specification Notation

1. Aerodynamic Forces and Moments

Using the non-dimensional coefficients, lift, drag and sideforce are calculated as follows:

$$L' = \left[C_{Lo} + C_{L\alpha}\alpha + C_Q Q\frac{c}{2V\tau} + C_{L\dot{\alpha}}\dot{\alpha}\frac{c}{2V\tau} + C_{L\delta e}\delta e\left[\frac{(V\tau + \Delta V\varepsilon)}{V\tau}\right]^2\right]\frac{\rho V\tau^2 S}{2} \qquad (2.3)$$

$$D = \left[C_{Do} + C_{D\alpha}\alpha + C_{D\delta e}\delta e\left[\frac{(V\tau + \Delta V\varepsilon)}{V\tau}\right]^2\right]\frac{\rho V\tau^2 S}{2} \qquad (2.4)$$

$$S_F = [C_{Y\beta}\beta + C_{Y\delta_r}\delta r]\frac{\rho V\tau^2 S}{2} \tag{2.5}$$

Once lift, drag and sideforce are calculated, these forces are translated into forces along the aircraft X, Y, and Z axes as shown in Equations 2.6 through 2.8. The terms F_{AX}, F_{AY} and F_{AZ} represent the resultant aerodynamic forces.

$$F_{AZ} = -L'\cos\alpha - D\sin\alpha \tag{2.6}$$

$$F_{AX} = L'\sin\alpha - D\cos\alpha - S_F\sin\beta \tag{2.7}$$

$$F_{AY} = S_F\cos\beta \tag{2.8}$$

The aerodynamic moments represent the torque forces about the center of the aircraft and are determined in the following equations

$$L_A = \left[C_{L\beta}\beta + C_{LP}P\frac{b}{2V\tau} + C_{LR}R\frac{b}{2V\tau} + C_{L\delta a}\delta a + C_{L\delta r}\delta r\right]\frac{\rho V \quad Sb}{} \tag{2.9}$$

$$M_A = \left[C_{Mo} + C_{M\alpha}\alpha + C_{MQ}Q\frac{c}{2V\tau} + C_{M\dot{\alpha}}\dot{\alpha}\frac{c}{2V\tau} + C_{M\delta e}\delta e\left[\frac{(V\tau + \Delta V\varepsilon)}{V\tau}\right]^2\right]\frac{\rho V\tau^2 Sc}{2} \tag{2.10}$$

$$N_A = \left[C_{N\beta}\beta + C_{NP}P\frac{b}{2V\tau} + C_{NR}R\frac{b}{2V\tau} + C_{N\delta a}\delta a + C_{N\delta r}\delta r\right]\frac{\rho V\tau^2 Sb}{2} \tag{2.11}$$

The forces and moments that result from the above calculations are added to other forces and moments at this time (Equations 2.12 - 2.17).

$$F_X = F_{AX} + F_{Thrust} \tag{2.12}$$

$$F_Y = F_{AY} \tag{2.13}$$

$$F_Z = F_{AZ} \tag{2.14}$$

$$L = L_A + L_{Torque} \tag{2.15}$$

$$M = M_A + M_{Thrust} + M_{Gyro} \tag{2.16}$$

$$N = N_A + N_{Thrust} + N_{Gyro} \tag{2.17}$$

Engine forces such thrust, torque and gyroscopic effect as well as environmental forces such as wind shear can have anywhere from a minor to significant effect on the forces and moments along all axes of the aircraft [Roskam, 1979]. However, in order to limit the complexity of the model, some simplifications are made. Engine thrust is limited to the X-axis only and no calculations are made for torque or gyroscopic effect. This can easily be included at a future date if desired and when these values are available.

2. Calculating Linear and Angular Acceleration

The total force equations are used to determine the linear acceleration of the aircraft [Nelson,1989]:

$$\dot{U} = VR - WQ - g\sin\theta + \frac{F_X}{m} \tag{2.18}$$

$$\dot{V} = WP - UR + g\sin\phi\cos\theta + \frac{F_Y}{m} \tag{2.19}$$

$$\dot{W} = UQ - VP + g\cos\phi\cos\theta + \frac{F_Z}{m} \tag{2.20}$$

The total moment equations are used to derive the equations for solving for angular acceleration:

$$L = I_{XX}\dot{P} - I_{XZ}\dot{R} - I_{XZ}PQ + (I_{ZZ} - I_{YY})RQ \tag{2.21}$$

$$M = I_{YY}\dot{Q} + (I_{XX} - I_{ZZ})PR + I_{XZ}(P^2 - R^2) \tag{2.22}$$

$$N = I_{ZZ}\dot{R} - I_{XZ}\dot{P} + (I_{YY} - I_{XX})PQ + I_{XZ}QR \tag{2.23}$$

However, prior to solving for either P or R, an interim step is required:

$$L'' = L + I_{XZ}PQ - (I_{ZZ} - I_{YY})RQ \tag{2.24}$$

$$N' = N - (I_{YY} - I_{XX})PQ - I_{XZ}RQ \tag{2.25}$$

Therefore,

$$\dot{P} = (L''I_{ZZ} - N'I_{XZ}) / (I_{XX}I_{ZZ} - I_{XZ}^2) \tag{2.26}$$

$$\dot{Q} = (M - (I_{XX} - I_{ZZ})PR - I_{XZ}(P^2 - R^2)) / I_{YY} \tag{2.27}$$

$$\dot{R} = (N'I_{XX} + L''I_{XZ}) / (I_{XX}I_{ZZ} - I_{XZ}^2) \tag{2.28}$$

are the equations for angular acceleration.

3. Calculating Linear and Angular Velocity

Linear and angular velocities are determined by numerically integrating the accelerations. The Trapezoidal Rule, sometimes referred to as the Modified Euler Method, is used. The general method for this integration technique is as follows:

$$P_n = P_{n-1} + \left(\frac{\dot{P}_{n-1} + \dot{P}_n}{2}\right)dt \tag{2.29}$$

where

P_n = new value of P

P_{n-1} = previous value of P

$\dot{P}_n$ = current rate-of-change of P (obtained from Euler prediction)

$\dot{P}_{n-1}$ = previous rate-of-change of P

dt = integration step size

4. Updating Position

Because position updates occur in the world coordinate system, a system must exist to convert the aircraft's linear velocities into changes in world position changes. Fortunately, this system was proposed years ago by Leonard Euler [Euler,1758]. By knowing the current attitude of the aircraft in world coordinates, the linear body velocities are easily converted into world rates by applying the following transformation, which effectively rotates the [U V W] vector by the Euler angles[Roskum,1980]. The order of transformation is important when utilizing this method and proceeds as follows:

$$\begin{bmatrix} U_{\phi} \\ V_{\phi} \\ W_{\phi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{2.30}$$

$$\begin{bmatrix} U_{\phi\theta} \\ V_{\phi\theta} \\ W_{\phi\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} U_{\phi} \\ V_{\phi} \\ W_{\phi} \end{bmatrix} \tag{2.31}$$

$$\begin{bmatrix} U_{W} \\ V_{W} \\ W_{W} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} U_{\phi\theta} \\ V_{\phi\theta} \\ W_{\phi\theta} \end{bmatrix} \tag{2.32}$$

Integrating the resultant velocity vector, now in world coordinates, by the time step of the program, a position update is obtained (Equation 2.33).

$$\begin{bmatrix} X_{W} \\ Y_{W} \\ Z_{W} \end{bmatrix} = \begin{bmatrix} X_{W_{old}} \\ Y_{W_{old}} \\ Z_{W_{old}} \end{bmatrix} + \begin{bmatrix} U_{W} \\ V_{W} \\ W_{W} \end{bmatrix} dt \tag{2.33}$$

How Euler angles are obtained is addressed in the next chapter.

III. METHODS OF ORIENTATION

A. INTRODUCTION

The aerodynamic model generates rotational velocities relative to the fixed aircraft body coordinate system. But, just as position updates could only be determined after converting linear velocities into world coordinates, orientation updates require conversion of angular velocities in a similar manner. Three methods exist for defining the conversion of angular velocities to orientations in world coordinates, each having its own particular advantages and disadvantages.

The most popular of these three methods is known as the Euler Method. Using a sequence of three angles, the Euler Method provides an intuitive description of aircraft attitude in world space [Rolfe,1986]. These angles consist of the familiar azimuth angle ψ, the pitch angle θ, and the bank angle ϕ. The next method, which has become popular in recent years, is the Quaternion Method. Based on the unit sphere, the Quaternion Method provides an elegant method of defining rotations through the use of four parameters. Three of the coordinates describe the axis of rotation while the fourth is determined by the angle through which the rotation occurs [Shoemake, 1985]. The third method of defining orientation is the Direction Cosine Matrix. The direction cosines relate the aircraft body axis frame to the world reference frame. Direction cosines, as used in flight simulation, can be determined from either Euler angles or quaternions, are utilized for transformations between axes, and, alternatively can be directly updated by incremental rotation matrices [Paul, 1981].

Each method has its own particular advantages and disadvantages and their use depends on the application and the implementation. Because NPSNET is a networked simulator, the orientation model used must not only render orientations in the world of the

aircraft being piloted, but also of other aircraft in the world either flying autonomously or piloted remotely across the network.

B. SURVEY OF METHODS

1. Euler Method

The most common method of defining an aircraft's orientation in world space is by the Euler attitude angles. Starting with the aircraft's axis origin aligned with the world's axis origin, the Euler angles specify three successive rotations to bring the world coordinates into alignment with the aircraft. The fact that there exist 24 possible ways to define rotations, each with potentially different results, means that the order of these rotations is important. Euler chose the convention of rotating first about the z axis, then about the new x axis and finally about the newest z axis. This convention exists in celestial mechanics, applied mechanics, and molecular and solid state physics. The convention used in quantum mechanics, nuclear physics, and particle physics, chooses to rotate first about the z, then the new y, and finally the new z [Burchfiel,1990]. The convention most often used in graphics is standard to aerospace engineers and has been proposed for use by DIS [UCF/IST 1990]. Using the right hand rule, rotations are made, first, counterclockwise about the z axis by the angle ψ, then counterclockwise about the new y axis by angle θ, and finally counterclockwise about the newest x axis by angle ϕ (Figure 3.1).

The range of values the attitude angles can take are:

$$\psi = \pm\pi \qquad \theta = \pm\frac{\pi}{2} \qquad \phi = \pm\pi$$

The aerodynamic model generates velocities in body coordinates. As seen in Equations 2.30-2.33, the linear body rates were transformed into world rates by application of the Euler angles. What follows is a method for obtaining these angles.

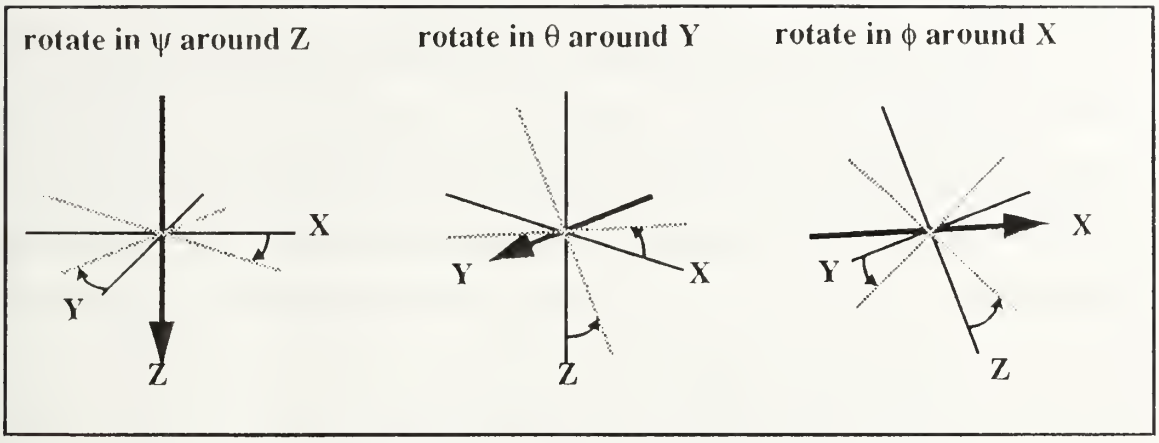

Figure 3.1: Euler Attitude Angle Rotation

There is a direct relationship between Euler attitude angles and the angular velocity of the aircraft around its body axes [Nelson,1989]. From this relationship (Equations 3.1-3.3), the rates of change of the attitude angles can be derived.

$$\dot{\phi} = P + Q\sin\phi\tan\theta + R\cos\phi\tan\theta \tag{3.1}$$

$$\dot{\theta} = Q\cos\phi - R\sin\phi \tag{3.2}$$

$$\dot{\psi} = Q\sin\phi\sec\theta + R\cos\phi\sec\theta \tag{3.3}$$

The inverse of the above equations are

$$P = \dot{\phi} - \dot{\psi}\sin\theta \tag{3.4}$$

$$Q = \dot{\theta}\cos\phi + \dot{\psi}\sin\phi\cos\theta \tag{3.5}$$

$$R = -\dot{\theta}\sin\phi + \dot{\psi}\cos\phi\cos\theta \tag{3.6}$$

Equations 3.1 through 3.3 are also known as the gimbal equations and are quite commonly used in simulation. However, a problem exists when pitch, θ, goes through the vertical. That is, where pitch becomes +/-(π/2). At that point $\dot{\phi}$ and $\dot{\psi}$ become undefined. Implementing a flight dynamics model capable of complete vertical maneuvering, necessitates "fixing" the code so a division by zero doesn't occur.

2. Direction Cosines

In the case of a flight simulation, transforming between body coordinates and world coordinates is done quite frequently. A convenient way to represent the transformation between two coordinate systems is with the direction cosines. Using matrix notation and the direction cosines (a, b, c), the transformation from body to world axes is expressed by:

$$\begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.7}$$

X, Y and Z represent vectors of any kind, such as force, velocity and acceleration. The inverse relationship, converting world coordinates to body coordinates is the transpose:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} \tag{3.8}$$

In terms of the Euler attitude angles, the direction cosines for the above transformations are shown in Figure (3.2)

$a_1 = \cos\theta\cos\psi$	$b_1 = \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi$	$c_1 = \cos\phi\sin\theta\cos\psi - \sin\phi\sin\psi$
$a_2 = \cos\theta\sin\psi$	$b_2 = \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi$	$c_2 = \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi$
$a_3 = -\sin\theta$	$b_3 = \sin\phi\cos\theta$	$c_3 = \cos\phi\cos\theta$

Figure 3.2: Direction Cosines in Terms of Euler Angles

It should be noted that as there are 12 ways in which Euler angles can be defined, there also exist more ways to define the direction cosines. They depend on the coordinate system utilized.

The direction cosines are absolutely necessary for transformations between coordinate systems, whether Euler angles or quaternions are used. Direction cosines were already used for transforming linear velocities in body coordinates to world coordinates (Equations 2.30-2.32). By multiplying those transformation matrices into one matrix, the result would be identical to the transformation matrix of Equation 3.7. By using direction cosines, the need for determining the intermediate velocities is eliminated.

3. Quaternion Method

An alternate method that has gained popularity in the graphics community in the mid 1980's is through the use of the unit quaternion. Not a new method, quaternions have been around for over a century. Augmenting the "Four-Parameter Method", they have been useful to aerodynamic engineers for some time and are still the method of choice for describing spacecraft orientation [Mitchell, 1965]. Discovered by Sir William Rowan Hamilton in 1843 as a result of a search for a successor to complex numbers, quaternions provide an efficient means for updating orientations [Shoemake, 1985].

a. Defining Quaternions

There are numerous ways to interpret the quaternion mathematically. The can be described as an algebraic quantity,

$$w + ix + jy + kz \tag{3.9}$$

as a point in three dimensional projective space (w, x, y, z), as a linear transformation of four space (Matrix), or as a scalar plus 3-vector:

$$(w, \vec{v}) \qquad \vec{v} = i\vec{x} + j\vec{y} + k\vec{z} \tag{3.10}$$

The best notation depends on their intended use. The most intuitive approach is to view the quaternion as a scalar plus 3-vector (Equation 3.10). However, for algebraic manipulation, Equation 3.9, generally becomes more useful.

A common way of defining quaternion orientation is in combination with Euler's Theorem which states that the orientation of a rigid body can be described as a rotation about an axis $\hat{v}$ by rotation angle Φ (Figure 3.3)[Goldstien,1980]. Constraining the vector part to be of unit magnitude, the quaternion becomes:

$$\cos\frac{\Phi}{2}, \hat{v}\sin\frac{\Phi}{2} \tag{3.11}$$

This representation is always of unit magnitude such that:

$$w^2 + x^2 + y^2 + z^2 = 1 \tag{3.12}$$

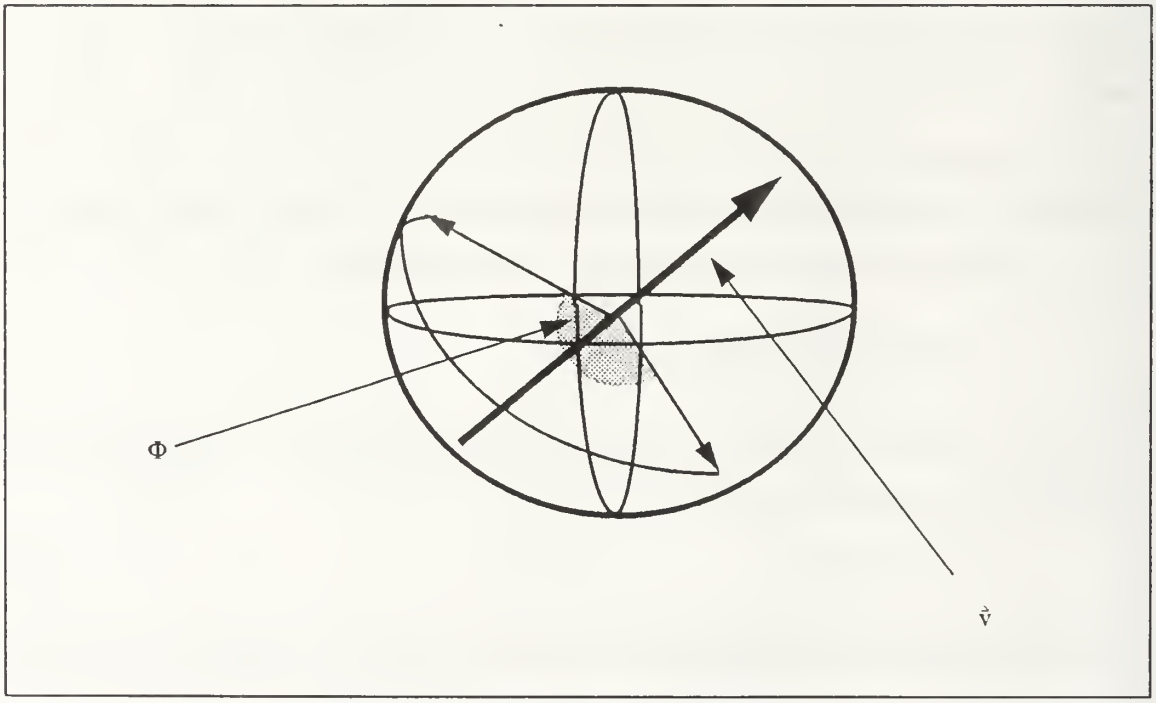

Figure 3.3: Quaternion Orientation

Prior to defining how to rotate a rigid body using the unit quaternion, it is necessary to review some of the mathematics associated with the quaternion. For the purpose of a flight simulation, an understanding of the multiplication and the quaternion derivative is necessary. More comprehensive reviews are obtainable from [Shoemake,1985; Golstein,1980].

b. Quaternion Multiplication

Almost every application involving quaternions makes use of the mathematics associated with their multiplication. Similar to the algebra associated with imaginary numbers, quaternions have three imaginary units, i, j, and k and are non-commutative under multiplication with

$$i^2 = j^2 = k^2 = -1 \tag{3.13}$$

where

$$ij = k = -ji \qquad jk = i = -kj \qquad ki = j = -ik \tag{3.14}$$

In algebraic notation, the product of quaternion Q multiplied by quaternion Q_1 is

$$\begin{aligned} QQ_1 &= (w + ix + jy + kz)\,(w_1 + ix_1 + jy_1 + kz_1) \\ &= (ww_1 - xx_1 - yy_1 - zz_1) \\ &+ i(xw_1 + wx_1 - zy_1 + yz_1) \\ &+ j(yw_1 + zx_1 - wy_1 + xz_1) \\ &+ k(zw_1 + yx_1 - xy_1 + wz_1) \end{aligned} \tag{3.15}$$

and is easily implemented in computer code. Vector notation is more intuitive to understand but not as simple to code:

$$QQ_1 = (w, \vec{v})\,(w_1, \vec{v}_1) = ww_1 - \vec{v} \cdot \vec{v}_1,\, w\vec{v}_1 + w_1\vec{v} + \vec{v} \times \vec{v}_1 \tag{3.16}$$

The result of the above multiplication is a rotation from the orientation represented by Q to the new cumulative orientation of Q and Q_1 in quaternion terms.

Multiplication provides a method of orientation extrapolation that can be of benefit in a networked simulation. If Q_1 represents a finite rotation based on an integral time step, and Q represents the cumulative rotation, then a repeated multiplying of Q by Q_1 will result in a smooth rotation across a series of update frames [Burchfiel,1990].

c. Quaternions in Terms of Direction Cosines

One frame of axes (body coordinates) can be brought into coincidence with a reference frame by a single rotation D about a fixed axis making angles A, B, and C with a second reference frame (world coordinates). The four parameters A, B, C, and D, therefore, define the orientation of the aircraft body in world coordinates [Rolfe,1986]. The transformation matrix relating the body to world coordinates using these four parameters is in Figure 3.4.

$$\begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} = \begin{bmatrix} 1-2\sin^2 A\sin^2\frac{1}{2}D & 2\cos A\cos B\sin^2\frac{1}{2}D - 2\cos C\sin\frac{1}{2}D\cos\frac{1}{2}D & 2\cos A\cos C\sin^2\frac{1}{2}D + 2\cos B\sin\frac{1}{2}D\cos\frac{1}{2}D \\ 2\cos A\cos B\sin^2\frac{1}{2}D + 2\cos C\cos\frac{1}{2}D\sin\frac{1}{2}D & 1-2\sin^2\frac{1}{2}D\sin^2\frac{1}{2}B & 2\cos B\cos C\sin^2\frac{1}{2}D - 2\cos A\cos\frac{1}{2}D\sin\frac{1}{2}D \\ 2\cos A\cos C\sin^2\frac{1}{2}D - 2\cos B\sin\frac{1}{2}D\cos\frac{1}{2}D & 2\cos B\cos C\sin^2\frac{1}{2}D + 2\cos A\sin\frac{1}{2}D\cos\frac{1}{2}D & 1-2\sin^2 C\sin^2\frac{1}{2}D \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Figure 3.4: Four Parameter Method

By substituting the following quaternion parameters, this method is further simplified:

$$\begin{aligned} q_0 &= \cos\frac{1}{2}D \\ q_1 &= \cos A\sin\frac{1}{2}D \\ q_2 &= \cos B\sin\frac{1}{2}D \end{aligned} \qquad \textbf{(3.17)}$$

$$q_3 = \cos C \sin \frac{1}{2} D$$

and the transformation matrix becomes,

$$\begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \textbf{(3.18)}$$

which represent the transform based on the unit quaternion. This result is used for converting determining position updates as well as orientation updates.

d. Derivative of the Unit Quaternion

To update the quaternion from angular accelerations, the following equations are used:

$$\begin{aligned} \dot{q}_0 &= -\frac{1}{2}(q_1P + q_2Q + q_3R) \\ \dot{q}_1 &= \frac{1}{2}(q_0P + q_2R - q_3Q) \\ \dot{q}_2 &= \frac{1}{2}(q_0Q + q_3P - q_1R) \\ \dot{q}_3 &= \frac{1}{2}(q_0R + q_1Q - q_2P) \end{aligned} \quad \textbf{(3.19)}$$

Because of the constraint that the quaternion be of unit value and assuming an integration step size of less than 1, the above set of equations become:

$$\begin{aligned} \dot{q}_0 &= -\frac{1}{2}(q_1P + q_2Q + q_3R) + \lambda q_0 \\ \dot{q}_1 &= \frac{1}{2}(q_0P + q_2R - q_3Q) + \lambda q_1 \\ \dot{q}_2 &= \frac{1}{2}(q_0Q + q_3P - q_1R) + \lambda q_2 \\ \dot{q}_3 &= \frac{1}{2}(q_0R + q_1Q - q_2P) + \lambda q_3 \end{aligned} \quad \textbf{(3.20)}$$

where

$$\lambda = 1 - (q_0^2 + q_1^2 + q_2^2 + q_3^2) \tag{3.20}$$

e. Euler Angles from Quaternions

Many of the auxiliary computations involved with a flight simulation require the use of Euler angles. To obtain these angles from the transformation matrix (Figure 3.2), the following method is utilized.

Because pitch is limited to $\pm\pi/2$, the $\cos(\theta)$ is always positive. As a result, obtaining these angles is relatively simple. The elevation angle is derived as follows:

$$\begin{aligned} \sin\theta &= -(a_3) \\ \theta &= \text{asin}\,(a_3) \end{aligned} \tag{3.21}$$

To obtain the azimuth (ψ) angle it must be noted that, since the $\cos(\theta)$ is always positive, the sign value of a_2 always reflects the sign value of the $\sin(\psi)$:

$$a_2 = \cos\theta\sin\psi \tag{3.22}$$

Therefore:

$$\psi = \text{acos}\left(\frac{a_1}{\cos\theta}\right) \cdot (\text{sign}\,[a_2]) \tag{3.23}$$

The roll (ϕ) angle is similarly obtained:

$$\phi = \text{acos}\left(\frac{c_3}{\cos\theta}\right) \cdot (\text{sign}\,[b_3]) \tag{3.24}$$

C. ADVANTAGES AND DISADVANTAGES

Quaternions and Euler angles each have their own advantages and disadvantages. Euler angles use only three components instead of four to represent orientation. If one were to send quaternions over a network in place of Euler angles, as has been proposed for DIS, network traffic would increase. However, in cases where angular rates remain constant for

long periods of time, by extrapolating orientation updates with quaternions, it would be necessary to send an update only when an angle rate change occurs [Burchfiel,1990].

Quaternions can be computed directly from the dynamic equations, bypassing the computation of transcendental functions necessary in computing Euler angles. If each transcendental function costs approximately 20 arithmetic operations then the net cost for deriving a rotation update using the Euler Method is 94 operations [Burchfiel,1990]. This can be compared to 42 operations using the quaternion method. In flight simulation, Euler angles are necessary for use in other simulator functions. This means that, approximately, another 64 operations are necessary, making the Euler Method slightly more computationally efficient. However, in many cases it is not necessary to compute the angles at each orientation update. Network updates should only occur when a rate change occurs. Additionally, radars, gyros and attitude indicators can be updated at slower rates. The bottom line is that orientations can be achieved very efficiently utilizing quaternions, without calculating Euler angles. When Euler angles are needed for other aircraft functions then quaternions become less efficient, depending on how often they need to be computed (Figure 3.5).

OPERATION	# EULER OPS	# QUATERNION OPS
DERIVATION	96	42
CREATING ROTATION MATRIX	20	32
TOTAL	116	74
EULER ANGLE CONVERSION	0	64
TOTAL CALCULATIONS	116	138

Figure 3.5: Efficiency Comparison of Euler and Quaternion Methods

The most significant advantage of quaternions is that no singularity exists when pitch angle (θ) passes through $\pm\pi/2$. In the Euler Method, $\dot{P}$ and $\dot{R}$ both become undefined due to division by zero. Techniques exist, however, for working around this singularity. Truncating values as $\pm\pi/2$ is approached will avoid this problem. If the pitch angle is truncated at values of 89.99 and 90.01 then a 0.02 degree rotation skip results [Goldiez,1991]. Depending on the speed of the program and the rotation rates desired, this may not be noticeable. However, in more fine grained simulations, where a slow vertical maneuver is executed, it is a factor.

D. PROPOSED ORIENTATION MODEL

The use of quaternions is desirable for many reasons. Since a dynamics model is used to drive the piloted aircraft, and the graphics platform utilizes transformation matrices for rendering, using the quaternion method offers an efficient method of converting body rates to orientations in world space.

Numerous aircraft operate autonomously within the prototype simulation that change very little in angular velocity. When this simulator is eventually networked to other workstations, quaternions will provide a way to foward interpolate the resulting changes in orientation on the remote computers, thereby eliminating the need for the continued transmission of update packets. If updates are eventually needed, the quaternion rates can quickly and easily be converted into Euler angles for transmission.

As the number of aircraft increase in the simulated world, the number of calculations necessary for orientation updates begins to multiply. Since only the currently piloted aircraft makes use of the Euler angles for additional simulation functions, calculating Euler angles is not necessary for all other aircraft in the simulation. Therefore, it becomes more efficient to utilize quaternions for defining these rotations, saving approximately 42 arithmetic operations per update per aircraft.

In the piloted aircraft, at the expense of 22 calculations per orientation update, a smooth rotation exhibiting no singularity about its axes during vertical maneuvers is possible. Future versions of this model will incorporate slow speed, fine grain functionality, so smooth changes in orientation about all axes is important and are best obtained using the quaternion.

IV. IMPLEMENTATION ISSUES

Aside from converting the mathematical model directly to code, other considerations must be addressed in the simulation. First, it must be modular enough to be easily incorporated into the existing NPSNET networked battlefield simulation trainer. NPSNET requires that it be operable from a Silicon Graphics workstation using a spaceball peripheral for flight control input, and provide update information for transmission to other workstations utilizing existing network protocols. Second, it must be able to simulate many different types of aircraft and allow interaction with a potentially large number other aircraft, either piloted from other workstations or operating autonomously. Third, for future use by aeronautical engineers, the system must be designed so that the flight characteristics, of whichever aircraft being flown, are based on an aircraft's dimensional characteristics and stability coefficients.

A. OVERALL SYSTEM LAYOUT

To satisfy the prototype's basic requirements, the overall structure of the system is designed as shown in (Figure 4.1). An aircraft data file makes it simple to create new

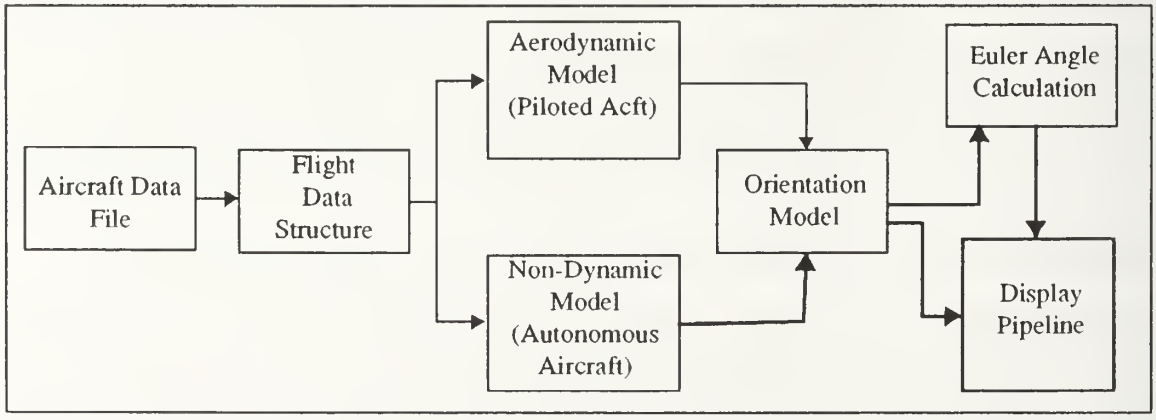

Figure 4.1: Prototype Flight Simulator Basic Structure

aircraft, position these aircraft, and designate their handling characteristics. It is also be used to initialize the flight parameters of the autonomous aircraft. The flight data structure contains information describing the current state of the aircraft and its design

specifications. The aerodynamic model is used for updating the piloted aircraft's body rates. The non-dynamic model also outputs a set of velocities, but unlike the dynamic model, it determines these velocities in accordance with a predetermined script designated by the implemented. The orientation model converts body rates to position and orientation in world space. Euler angles are then determined for the piloted aircraft and are needed to update cockpit displays. Autonomous aircraft do not require the calculation of Euler angles and bypass this function.

B. THE AIRCRAFT DATA INPUT FILE

Data records within the input file are divided into two categories, flight records and aircraft specification records. The flight record contains information describing the position of an aircraft and its initial flight parameters (Figure 4.2). The specification record contains the dimensional characteristics and stability coefficients describing a particular aircraft. By manipulating the data in the specification record, one can change an aircraft's basic design as desired. To link an aircraft to a particular set of specification data, all that is necessary is to match the "type aircraft" information within the two records. This system allows one specification record to be used for an unlimited number of flight records.

C. THE FLIGHT DATA STRUCTURE

The flight data structure provides a global source of information on the state of each aircraft in the simulation. The information contained here is necessary for operation of the aerodynamic and orientation models, and for updating cockpit displays (Figure 4.3). Note that the fourth item in this structure is a pointer to another data structure containing aircraft specification data. Not shown, this structure contains information identical to that found in the specification record of the data input file (Figure 4.2). Maintaining flight information in two separate files saves some storage space by allowing more than one aircraft use a single set of specification data.

Flight Record	
1	/id number/
1	/type aircraft/
200.0	/airspeed/
-950.0	/posx/
300.0	/altitude/
0.0	/posz/
0.0	/heading/
0.0	/initial angle of bank/
3.0	/initial gforce/
1	/status:0-piloted,1-levelturn,2-g controlled turn/
Specification Record	
4	/type aircraft-- A4/
1	/jet or prop jet:1 prop:0/
27.5	/b/
260.0	/S/
10.8	/c/
546.0	/m/
8090.0	/Ix/
25900.0	/Iy/
29200.0	/Iz/
1300.0	/Ixz/
0000.0	/max thrust/
8000.0	/mil thrust or horsepower/
0.03 0.3	/CDo /CDa
0.28 3.45 0.0 0.72 0.36	/CLo /CLa /CLq /CLda /CLde
0.0 -3.6 -0.38 -1.1 -0.5	/CMo /CMq /CMa /CMda /CMde
-0.98 0.17	/CYb /CYdr
-0.12 -0.26 0.14 0.08 -0.105	/CLb /CLp /CLr /CLda /CLdr
0.25 0.022 -0.35 0.06 0.032	/CNb /CNp /CNr /CNda /CNdr
0.2618 -0.5236 0.5236	/deflection limits of rud, ail, elevator (radians)/

Figure 4.2: Data Input Records

```
int       ID;              --number assignment
int       type;            --aircraft type
int       status;          --piloted:0 or autonomous level turn:1 climbing turn:2
typeptr   Tptr;            --pointer to aircraft specification data structure
float     Forces[3];       --forces in X,Y,Z dir
float     Torques[3];      --torques around X,Y,Z axis
float     linear_vel[3] :  --velocity in X,Y,Z direction
float     angular_vel[3];  --angular velocity around X,Y,Z axes
float     linear_accel[3]; --linear accelerations
float     angular_accel[3]; --angular accelerations
float     sideslip;        --sideslip or beta angle
float     ang_atk;         --angle of attack
float     d_ang_atk;       --angle of attack rate
float     lift;            --total lift
float     drag;            --total drag
double    Q[4];            --quaternion
Matrix    H;               --direction cosine matrix
float     euler_angles[3]; --euler angles angles in radians - yaw,pitch,roll
float     pos[3];          --world position in X, Y, Z
float     ref[3];          --look direction
float     vel_world[3];    --velocities in world position - X,Y,Z
float     gfor;            --amount of g force
float     rpm;             --engine rpm
float     elev;            --flight control positions
float     eltrim;          --elevator trim
float     rud;             --rudder position
float     ail;             --aileron position
float     thro;            --throttle position
int       flaps;           --flap position
int       gear;            --landing gear position
```

Figure 4.3: Aircraft Flight Data Structure

D. FLIGHT CONTROL INPUT

1. Throttle

The throttle in an aircraft is the pilot's primary means to control the engine. As such, a mapping of throttle position to engine rpm must be devised that incorporates delays associated with engine spool-up characteristics. In large, high speed simulators, rpm and engine data is retrieved from engine-specific lookup tables. Because tables such as these

are engine specific, a simpler, generic method was devised. By keeping track of the last throttle position, the following formula is utilized:

$$\Delta rpm = (T_{old} - T_{new})\,\varphi dt \qquad \textbf{(4.1)}$$

where

T = throttle position
dt = delta time
φ = engine spool-up delay factor

Since applications using this simulator include both jet and propeller aircraft, it was decided that allowances should be made for the differing characteristics of the two types of engines. Jet engines are generally rated in terms of thrust (lbs), while propellers are rated in terms of horsepower (ft-lbs/s) [Anderson, 1989]. Since the aerodynamic model uses thrust in terms of lbs, it can use the data for jets directly. However, propeller driven aircraft require the following conversion:

$$T = \frac{550\eta HP}{V\tau}\frac{\rho}{\rho_0} \qquad \textbf{(4.2)}$$

where

η = propeller efficiency (usually around .8)
HP = engine rated horsepower
$\frac{\rho}{\rho_0}$ = density altitude ratio where ρ_o is the density at sea level

2. Aileron, Elevator and Rudder Control

A normal aircraft stick exhibits two degrees of freedom, left-right for aileron control and back-forward for elevator control. Therefore, control inputs from the spaceball were limited to these directions. The maximum deflection of the control stick is information entered via the specification records. It is a simple procedure to read deflection data from the spaceball and linearly map it to a control deflection somewhere between +/- max obtainable deflection.

Rudder input is handled in a similar manner to aileron and elevator control. Maximum rudder deflection is entered via the specification records so that its position can be linearly determined from the amount of rudder control input. Rudder input is via the keyboard using the left and right arrow keys. Although not the same as using rudder pedals, separating the rudder input from the control stick inputs forces the operator to be concerned with the issue of rudder and aileron coordination.

3. CONTROL OF NON-PILOTED AIRCRAFT

In the current system, an autonomous aircraft can exhibit one of two behaviors, depending on the value of the "status" data in the flight data record. The first behavior is that of a level turn. The initial roll angle, also a part of the flight data record (degrees) provides the seed for calculating the desired body rates. From the bank angle, the amount of g force necessary to maintain level flight at this angle of bank is [Anderson, 1989]:

$$\text{G-force} = \frac{1}{\cos\phi} \tag{4.3}$$

from the g-force, the rate of turn (ROT) in degrees/sec can be determined:

$$\text{ROT} = \frac{g\sqrt{\text{G-force}^2 - 1}}{V_\tau} \tag{4.4}$$

where g is gravity (32.2 ft/sec^2). A level turn requires a balance between the angular velocities Q, pitch rate, and R, yaw rate. Therefore, from Equations 3.4-3.6:

$$\begin{aligned} Q &= \text{ROT}\sin\phi \\ R &= \text{ROT}\cos\phi \end{aligned} \tag{4.5}$$

The second behavior is that of a non-level turn. This turn is based on the initial angle of bank and g force entered in the data input file. The yaw rate, R, is calculated the same as for a level turn. The pitch rate, Q, is also computed using these formulas but, instead of using the computed g force for a level turn, it is calculated using the amount of desired g force entered via the data file. The calculations necessary to compute the angular

velocities are completed during program initialization and do not impact on the run time of the simulation.

E. SPEED OF AERODYNAMIC MODEL

The aerodynamic model exhibits a tendency to "blow up" if the time step between calculations becomes too great. This becomes evident when the aircraft speed and rotation rates increase. The solution to this problem is to run the aerodynamic model at a faster rate than the rest of the system. The trick is to determine how fast. "Smart" integration schemes can be found in the literature that increase or decrease the number of time steps according to the amount of numerical divergence present in the integrated values [Press,et al,1990].

A simple method is used in this program to solve for this problem. Because there exists a direct relationship between an aircraft's speed and its tendency to produce a numerical instability, the time step was adjusted in direct relationship with the aircraft's airspeed. Prior to updating body rates in the aerodynamic model, aircraft airspeed is measured and a the number of time steps is calculated (Figure 4.4). The timestep factor used in the current program is 0.03 seconds.

```
aerodynamic model
        read aircraft velocity
        time step for aero model = timestep_factor * velocity
        for computed_time_step loop
                do aero calculations
                update aircraft state variables
        end loop
exit aerodynamic model
```

Figure 4.4: Algorithm for Computing Time Step

Increasing the time step does not decrease performance of the overall system. The speed of the aerodynamic model in the current implementation ranges from 17.6 ms to 18.8 ms for time steps ranging from two to 160. Running at approximately 120 ms, the graphics pipeline definitely remains the limiting factor in this system.

V. CONCLUSIONS AND FUTURE WORK

The techniques presented in this paper have proven to be an effective method of implementing a graphical dynamic flight simulation on a matrix-based graphics computer in real-time. Like most research and academic projects, this aircraft simulator is structured to allow for the addition of more detailed functionality. Current work includes the integration of a weapons delivery system and avionics suite.

The orientation model functions developed during the course of this paper will become part of the C program library at the school, thereby providing an alternate and more flexible tool for manipulating solid objects in a graphical environment. Integration of this orientation model into other dynamic simulation systems is also under investigation.

APPENDIX A

```
/*************************************************************************/
/* FILE: aero.c                                                          */
/* AUTHOR: Joe Cooke                                                     */
/* DATE: 1/9/92                                                          */
/* DESCRIPTION: This file contains the mathematical model for all        */
/* aerodynamic calculations.                                             */
/* Input: aircraft state structure(P)                                    */
/*        aircraft specification stucture(T)                             */
/* Output:updated state structure(P)                                     */
/*************************************************************************/
aero_model(acftptr P,typeptr T,float deltatime)
{
float Q,QS,QSc,QSb,c2u,b2u,g,w,m;          /*misc. variables*/
float tot_vel;                             /*total velocity*/
float densaltratio,densalt;                /*atmosperic data*/
float aoa, cos_aoa, sin_aoa;               /*angle of attack data*/
float cos_sideslip, sin_sideslip;          /*sideslip data*/
float For_thrust_X;                        /*thrust force*/
float ailpos,rudpos, elevpos;              /*control positions*/
float Itemp, Ltemp, Ntemp;                 /*temporary terms*/
float du, dv, dw, dp, dq, dr;              /*newly computed accels*/
float diff,                                /*temp var for rpm determination*/
float loopdelta;                           /*deltatime / loop iterations*/
float vworld[3],vwor[3]l                   /*vel in world inertial frame*/
float vaircraft[3],                        /*vel in aicraft frame*/
float hpower;                              /*horsepower*/
float max_thrust;                          /*thrust*/
float loop_iterations;                     /*number of aero calc iterations*/
int ixt, alt;                              /*index and loop vars*/
Matrix A_MATRIX;                           /*temporary matrix*/

 /*****************test for valid structure*********************/
 if (P == NULL || T == NULL){
    printf("****error: null pointer sent to aero.c****\n\n");
    return;
    }
 /****some initial assignments****/
 m = T->m;
 g = GRAVITY;
 w = m * GRAVITY;
 P->gfor = P->lift/w; /*calculate current G's*/
 if (P->u < 10.0) P->u = 10.0;
 tot_vel = fsqrt(P->v*P->v + P->w*P->w + P->u*P->u);

 /************atmosheric density calculations*****************/
 if (P->posy >= 0.0 && P->posy < MAX_ALT){
    alt = (int) (P->posy/(ALT_INCREMENT*FT_TO_METERS));
    densalt = densaltarray[alt];
    densaltratio = densalt/DENSITYSL;
    }
 else{
    densalt = DENSITYSL;
    densaltratio = densalt/DENSITYSL;
    }

 /***************calculate new rpm*****************************/
 diff = (P->thro - P->rpm)/(2.0/deltatime);
 P->rpm = P->rpm + diff;

 /***************calculate new thrust**************************/
 if (T->class == 0){ /*if a prop plane*/
    hpower = PROP_EFFIC * T->horsepwr;
    max_thrust = (hpower * HPR2THRUST)/ tot_vel;
```

```
    For_thrust_X = max_thrust * P->rpm * densaltratio;
    }
else{ /*if a jet plane*/
    if (T->max > T->mil) For_thrust_X = (T->max*(P->rpm)) * densaltratio;
    else For_thrust_X = (T->mil*(P->rpm)) * densaltratio;
    }

/**********determine control differentials*******************/
ailpos = (P->ail * T->def_ail);
rudpos = (P->rud * T->def_rud);
elevpos = (P->elev * T->def_elev) + P->eltrim;

/************adjust time step for aero calculations********/
loop_iterations = (float)((int)(tot_vel*TENTH_OF_AIRSPEED_FACTOR));
loopdelta = deltatime * 1.0/loop_iterations;

/***********loop to reduce time step for dynamic model*****/
for(ixt = 0;ixt<((int)loop_iterations);ixt ++){
    if (P->u < 10.0) P->u = 10.0;
    tot_vel = fsqrt(P->v*P->v + P->w*P->w + P->u*P->u);

    P->sideslip = fatan((P->v/P->u));
    sin_sideslip= (sin(P->sideslip));
    cos_sideslip= (cos(P->sideslip));

    aoa = fatan((P->w/P->u));/* + (5.0/RAD_TO_DEG)*/
    cos_aoa = (fcos(aoa));
    sin_aoa = (fsin(aoa));
    P->d_ang_atk = aoa - P->ang_atk;
    P->ang_atk = aoa;

    Q = 0.5*densalt*tot_vel*tot_vel;
    QS = Q *T->S;
    QSc = QS*T->c;
    QSb = QS*T->b;
    c2u = T->c/(2.0*tot_vel);
    b2u = T->b/(2.0*tot_vel);

    /**********calculating aerodynamic forces*********************/
    P->lift = (T->CLo +
    T->CLa * P->ang_atk +
    T->CLq * P->q * c2u +
    T->CLdalpha * P->d_ang_atk * c2u +
    T->CLde * elevpos ) * QS;

    P->drag = (T->CDo +
    T->CDa * P->ang_atk ) * QS;

    P->sideforce = (T->CYb * P->sideslip +
    T->CYdr * rudpos ) * QS;

    P->Fx = (P->lift * sin_aoa -
    P->drag * cos_aoa -
    P->sideforce * sin_sideslip);
    P->Fy = (P->sideforce * cos_sideslip);
    P->Fz = (-P->lift * cos_aoa - P->drag * sin_aoa);

    /**********calculating aerodynamic moments*******************/
    P->L = (T->CLb * P->sideslip +
    T->CLp * P->p * b2u +
    T->CLr * P->r * b2u +
    T->CLda * ailpos +
    T->CLdr * rudpos ) * QSb;

    P->M = (T->CMo +
    T->CMa * P->ang_atk +
    T->CMq * P->q * c2u +
    T->CMda * P->d_ang_atk * c2u +
    T->CMde * elevpos ) * QSc;

    P->N = (T->CNb * P->sideslip +
    T->CNp * P->p * b2u +
```

```
    T->CNr * P->r * b2u +
    T->CNda * ailpos +
    T->CNdr * rudpos ) * QSb;

    /******determine total moments and forces**********************/
    P->Fx = P->Fx + For_thrust_X;

    /*****temporary calculations*************************************************/
    Ltemp = P->L + T->Ixz * P->p * P->q - (T->Iz - T->Iy) * P->r * P->q;
    Ntemp = P->N - (T->Iy - T->Ix) * P->p * P->q - T->Ixz * P->r * P->q;
    Itemp = (T->Ix * T->Iz - T->Ixz * T->Ixz);

    /******determine Linear and Angular Accelerations*************************/
    du = (P->v * P->r - P->w * P->q + P->Fx/m - g * P->sptch );
    dv = (P->p * P->w - P->r * P->u + P->Fy/m + g * P->sroll * P->cptch);
    dw = (P->u * P->q - P->v * P->p + P->Fz/m + g * P->croll * P->cptch);
    dp = (Ltemp * T->Iz + Ntemp * T->Ixz) / Itemp;
    dq = (P->M - ((T->Ix - T->Iz) * P->p * P->r)
            - ((P->p * P->p - P->r * P->r) * T->Ixz)) / T->Iy;
    dr = (Ntemp * T->Ix + Ltemp * T->Ixz) / Itemp;

    /***determine velocities with trapizoidal integration****/
    P->u = P->u + (( P->udot + du )/2.0)*loopdelta;
    P->v = P->v + (( P->vdot + dv )/2.0)*loopdelta;
    P->w = P->w + (( P->wdot + dw )/2.0)*loopdelta;
    P->p = P->p + (( P->pdot + dp )/2.0)*loopdelta;
    P->q = P->q + (( P->qdot + dq )/2.0)*loopdelta;
    P->r = P->r + (( P->rdot + dr )/2.0)*loopdelta;

    /*********record last acceleration**********************/
    P->udot = du; P->vdot = dv; P->wdot = dw;
    P->pdot = dp; P->qdot = dq;P->rdot = dr;
    } /***end loop***/

/******incrementally rotate quaternion*********************/
increment_quat(P->p,P->q,P->r,P->Q,deltatime);

/*****update rotation matrix from quaternion***************/
rotation_matrix_from_quat(P->Q,P->H);

/*****extract euler angles from matrix*********************/
euler_angles_from_matrix(&P->sroll,&P->croll,&P->sptch,&P->cptch,
                         &P->shdg, &P->chdg,P->h_matrix);
euler_ang_frm_rot_matrix(&P->roll,&P->ptch,&P->hdg,P->H);
P->ptch = P->ptch * RAD_TO_DEG;
P->roll = P->roll * RAD_TO_DEG;
P->hdg = P->hdg * RAD_TO_DEG;

/****calculate new world velocity**************************/
vaircraft[0] = P->u;
vaircraft[1] = P->v;
vaircraft[2] = P->w;
transpose_matrix(P->H,A_MATRIX);
post_mult_matrix_by_vector(A_MATRIX,vaircraft,vworld);

vwor[0] = vworld[0] * FT_TO_METERS;
vwor[1] = -vworld[2] * FT_TO_METERS;
vwor[2] = vworld[1] * FT_TO_METERS;

/****calculate new world position**************************/
P->H[3][0] = P->posx += vwor[0] * deltatime;
P->H[3][1] = P->posy += vwor[1] * deltatime;
P->H[3][2] = P->posz += vwor[2] * deltatime;
P->H[3][3] = 1.0;

/********calculate world reference point*****************/
P->refx = P->posx + (P->cptch * P->chdg);
P->refz = P->posz + (P->cptch * P->shdg);
P->refy = P->posy + P->sptch;
}
```

APPENDIX B

```
/*************************************************************************/
/* FILE: quaternions.c                                                   */
/* AUTHOR: Joe Cooke                                                     */
/* DATE: 1/9/92                                                          */
/* DESCRIPTION: This file contains the functions for the quaternion      */
/* rotation and matrix operations                                        */
/*************************************************************************/

/***multiply quaternion Qr by Ql and return the product in Qr***/
mult_quaternion(Ql,Qr)
    double Ql[4],Qr[4];
{
    double n_factor,q;

n_factor= 1.0/((Qr[0]*Qr[0])+(Qr[1]*Qr[1])+(Qr[2]*Qr[2])+(Qr[3]*Qr[3]));

Qr[0]=n_factor*(Ql[0]*Qr[0] - Ql[1]*Qr[1] - Ql[2]*Qr[2] - Ql[3]*Qr[3]);
Qr[1]=n_factor*(Ql[1]*Qr[0] + Ql[0]*Qr[1] + Ql[2]*Qr[3] - Ql[3]*Qr[2]);
Qr[2]=n_factor*(Ql[2]*Qr[0] + Ql[0]*Qr[2] + Ql[3]*Qr[1] - Ql[1]*Qr[3]);
Qr[3]=n_factor*(Ql[3]*Qr[0] + Ql[0]*Qr[3] + Ql[1]*Qr[2] - Ql[2]*Qr[1]);
}


/***convert roll, pitch and yaw angles to a single quaternion*****/
/***useful for intitalizing a quaternion with initial orientation***/
euler_to_quaternion(roll,pitch,yaw,Q)
    float roll,pitch,yaw;
    double Q[4];
{
    double r,p,y,cosr,cosp,cosy,sinr,sinp,siny;

r = (((double)roll/2.0)/RAD_TO_DEG);
p = (double)((pitch/2.0)/RAD_TO_DEG);
y = (double)((yaw/2.0)/RAD_TO_DEG);
cosr = cos(r); cosp = cos(p); cosy = cos(y);
sinr = sin(r); sinp = sin(p); siny = sin(y);
Q[0] = (cosr*cosp*cosy)+(sinr*sinp*siny);
Q[1] = (sinr*cosp*cosy)-(cosr*sinp*siny);
Q[2] = (cosr*sinp*cosy)+(sinr*cosp*siny);
Q[3] =(-sinr*sinp*cosy)+(cosr*cosp*siny);
 }


/***increment a quaternion from incremental body angular rates******/
/***p=roll q=pitch and r=yaw deltas*****************************/
increment_quat(p,q,r,Q,deltatime)
    float p,q,r;
    double Q[4];
    float deltatime;
{
    double factor,qq1,qq2,qq3,qq0,dq0,dq1,dq2,dq3,p2,q2,r2;

qq0 = Q[0]*Q[0];
qq1 = Q[1]*Q[1];
qq2 = Q[2]*Q[2];
qq3 = Q[3]*Q[3];

p2 = (double)(p*0.5*deltatime);
q2 = (double)(q*0.5*deltatime);
r2 = (double)(r*0.5*deltatime);
factor = 1.0-(qq0+qq1+qq2+qq3);

dq0 = -(Q[1]*p2 + Q[2]*q2 + Q[3]*r2) + Q[0]*factor;
```

```
dq1 = (Q[0]*p2 + Q[2]*r2 - Q[3]*q2) + Q[1]*factor;
dq2 = (Q[0]*q2 + Q[3]*p2 - Q[1]*r2) + Q[2]*factor;
dq3 = (Q[0]*r2 + Q[1]*q2 - Q[2]*p2) + Q[3]*factor;
Q[0] = Q[0] + dq0;
Q[1] = Q[1] + dq1;
Q[2] = Q[2] + dq2;
Q[3] = Q[3] + dq3;
}


/***matrix for conversion from worldrates to body rates***/

rotation_matrix_from_quat(Q,M)
    double Q[4];
    Matrix M;
{
    double f,wf,xf,yf,zf,xxf,yyf,zzf,xyf,xzf,yzf,wxf,wyf,wzf;

f = 2.0/(Q[0]*Q[0]+Q[1]*Q[1]+Q[2]*Q[2]+Q[3]*Q[3]);

xf = f * Q[1];
yf = f * Q[2];
zf = f * Q[3];
xxf = xf*Q[1];
yyf = yf*Q[2];
zzf = zf*Q[3];
xyf = xf*Q[2];
xzf = xf*Q[3];
yzf = yf*Q[3];
wxf = xf*Q[0];
wyf = yf*Q[0];
wzf = zf*Q[0];
M[0][0] = (float)(1.0-(yyf+zzf));
M[0][1] = (float)(wzf+xyf);
M[0][2] = (float)(xzf-wyf);
M[1][0] = (float)(xyf-wzf);
M[1][1] = (float)(1.0-(xxf+zzf));
M[1][2] = (float)(yzf+wxf);
M[2][0] = (float)(xzf+wyf);
M[2][1] = (float)(yzf-wxf);
M[2][2] = (float)(1.0-(xxf+yyf));
M[3][0] = 0.; M[3][1] = 0.; M[3][2] = 0.; M[3][3] = 1.;
}

/***this function "extracts" sine and cosine data from a  cosine matrix
    represented in conventional form.*********************************/
euler_angles_from_matrix(sroll,croll,sptch,cptch,shdg,chdg,M)
    float *sroll,*croll,*sptch,*cptch,*shdg,*chdg;
    Matrix M;
{
     float sp,cp,sr,cr,sy,cy;
sp = -M[0][2];                          /*sinpitch*/
cp = fsqrt(1.0-(sp*sp));                /*cospitch*/
if (cp == 0.0){                         /*if pitch at +/- 90deg*/
    sy  = 0.0;                         /*sinyaw */
    cy  = 1.0;                         /*cosyaw */
    sr  = M[1][0];                     /*sinroll*/
    cr  = M[1][1];                     /*cosroll*/
    }
else{
    sy  = M[0][1]/(cp);              /*sinyaw  */
    cy  = M[0][0]/(cp);              /*cosyaw  */
    sr  = M[1][2]/(cp);              /*sinroll */
    cr  = M[2][2]/(cp);              /*cosroll */
    }
   *sptch = sp;*cptch = cp;*sroll = sr;*croll = cr;*shdg = sy;*chdg = cy;
}
```

```
/***The following function will return the roll, pitch and yaw from a standard
transformation matrix.This routine should work for a transformation in a coordinate
system from body coords where the x y and z axes move in pitch roll and yaw
respectively, to world coordinates where X Y and Z point North, East and Down
respectively.*************************************************************/

euler_ang_frm_rot_matrix(roll,pitch,yaw,matrix)
    float *roll,*pitch,*yaw;
    Matrix matrix;
 {
    float cospitch,sinpitch,roll_fact,yaw_fact;

 /**determine pitch angle in radians and cospitch**/
 sinpitch = -matrix[0][2];
 cospitch = fsqrt(1.0 - (sinpitch * sinpitch));
 *pitch = -(fasin(matrix[0][2]));

 /**determine roll angle in radians**/
 roll_fact = matrix[2][2]/cospitch;
 if (cospitch != 0.0)
    {
    if (matrix[1][2] < 0.0) *roll = -(facos(roll_fact));
    else
        *roll = facos(roll_fact);
    }
 else
    *roll = 0.0;

 /**determine yaw angle in radians**/
 yaw_fact = matrix[0][0]/cospitch;
 if (cospitch != 0.0)
    {
    if (matrix[1][0] < 0.0) *yaw = -(facos(yaw_fact));
    else
        *yaw = facos(yaw_fact);
    }
 else
    *yaw = 0.0;

/***this routine assumes matrix is in graphics convention form. The robotics convention
requires an additional matrix transposition prior to utilizing this function*********/

euler_to_sgi_axis_convert(M_matrix,W_matrix)
    Matrix M_matrix, W_matrix;
{
W_matrix[0][0] = M_matrix[0][0];
W_matrix[1][0] = M_matrix[1][0];
W_matrix[2][0] = M_matrix[2][0];
W_matrix[0][1] = -M_matrix[0][2];
W_matrix[1][1] = -M_matrix[1][2];
W_matrix[2][1] = -M_matrix[2][2];
W_matrix[0][2] = M_matrix[0][1];
W_matrix[1][2] = M_matrix[1][1];
W_matrix[2][2] = M_matrix[2][1];
W_matrix[3][0] = M_matrix[3][0];
W_matrix[3][1] = M_matrix[3][1];
W_matrix[3][2] = M_matrix[3][2];
W_matrix[3][3] = 1.0;
}

/***Returns a vecter result*********************************************************/
void post_mult_matrix_by_vector(Matrix M_matrix,float * V_array, float * Result)
 {
 Result[0] = V_array[0]*M_matrix[0][0] +
 V_array[1]*M_matrix[0][1] + V_array[2]*M_matrix[0][2];
 Result[1] = V_array[0]*M_matrix[1][0] +
 V_array[1]*M_matrix[1][1] + V_array[2]*M_matrix[1][2];
 Result[2] = V_array[0]*M_matrix[2][0] +
 V_array[1]*M_matrix[2][1] + V_array[2]*M_matrix[2][2];
 }
```

APPENDIX C

A. INTEGRATION OF AERODYNAMICS INTO NPSNET

To complete the integration of the aerodynamic model into NPSNET the following steps must be accomplished:

- •Modify the vehicle model structure to fully describe a dynamic aircraft.
- •Create a method to assign aircraft characteristics on program initialization.
- •Add cockpit display as the appropriate user interface.
- •Modify input devices for aircraft control (simulate aircraft control stick).
- •Add aerodynamic model for control of the piloted aircraft.
- •Add orientation model based on quaternions for aircraft rotation.

B. NEW VEHICLE MODEL STRUCTURE

The current NPSNET has one model for all types of vehicles. Tanks, helicopters, jets and jeeps all move under the same sets of rules, and only one basic data structure for all vehicles was previously created. Because air vehicles operate in a different dynamic environment than ground vehicles, it is necessary to create an additional data structure for their description. A logical way to add additional data for an aircraft vehicle is through the use of a union data structure. Imbedding a union data structure containing the new aerodynamic model within the original data structure allows the insertion of additional data needed for a more complex model, without having to re-engineer the entire program with a completely new data structure (Figure A3.1).

```
...(attached to Old Data Structure For Vehicle Model)
union{
        DEFVEH defaultveh;
        AIRVEH airveh;
        GNDVEH gndveh;
        SHIPVEH shipveh;
        SIMNETVEH simtype;
        } veh;
```

Figure C.1: Union to Be Inserted into Current Vehicle Data Structure

Two data structures are added to handle the aerodynamic model. Figure A3.2 shows the structure containing the vehicle state information and Figure A3.3 shows the structure containing the vehicle specifications. To connect the two structures, the pointer, TYPEPTR, was created.

```
typedef FLIGHTSPECS *   TYPEPTR;
typedef struct
        {
        int    totalacft;              /*track total number of aircraft in simul*/
        TYPEPTR Tptr;                  /*pointer to aircraft type information*/
        float      forc[3];            /*lbs: Forces in X,Y,Z direction*/
        float      torq[3];            /*ft-lbs: Torques around X,Y,Z axis*/
        float      lin_vel[3];         /*ft/sec: linear vel(u,v,w) in X,Y,Z dir*/
        float      ang_vel[3];         /*rad/sec: ang vel(p,q,r) around X,Y,Z axes*/
        float      lin_accel[3];       /*fps2: linear accels in X,Y,Z direction*/
        float      ang_accel[3];       /*deg/sec2:angaccel(p,q,r) around X,Y,Z axes*/
        float      sideslip;           /*sideslip or beta*/
        float       ang_atk;           /*angle of attack*/
        float      d_ang_atk;          /*angle of attack rate*/
        float      lift;               /*total lift*/
        float      drag;               /*total drag*/
        double     quat[4];            /*quaternion*/
        Matrix     h_matrix;           /*direction cosine matrix of world position*/
        float      sroll;              /*sine of roll*/
        float      croll;              /*cosine of roll*/
        float      sptch;              /*sine of pitch*/
        float      cptch;              /*cosine of pitch*/
        float      shdg;               /*sine of heading*/
        float      chdg;               /*cosine of heading*/
        float       euler_angles[3];   /*euler angles in radians - roll,pitch,yaw*/
        float       vel_world[3];      /*meters/sec: velocity in SGI coords X,Y,Z*/
        float      hdg;                /*world heading*/
        float      gfor;               /*g force exerted from back stick*/
        float      rpm;                /*fraction: 0.0  1.0 engine rpm*/
        float      elev;               /*position: -1.0 to 1.0*/
        float      eltrim;             /*elevator trim*/
        float      rud;                /*position: -1.0 to 1.0*/
        float      ail;                /*position: -1.0 to 1.0*/
        float      thro;               /*position: 0.0 to 1.0*/
        int        flaps;              /*boolean: 0-flapsup 1-flapsdown*/
        int        gear;               /*boolean: 0-gearup 1-geardown*/
        } AIRVEH;
```

Figure C.2: Data Stucture Containing Aircraft State Information

```
typedef struct
        {
        int     type;
        int     class;          /*1:jet 0:prop*/
        float   span;           /*wing span  */
        float   sarea;          /*wing area  */
        float   cord;           /*mean aerodynamic cord*/
        float   mass;           /*mass       */
        float   Ix;             /*x axis inertial term*/
        float   Iy;             /*y axis inertial term*/
        float   Iz;             /*z axis inertial term*/
        float   Ixz;            /*cross inertial term*/
        float   horsepwr;       /*used for props*/
        float   max;            /*max afterburner thrust*/
        float   mil;            /*max non afterburner thrust*/
        float   CDo;            /*reference drag coefficient          */
        float   CDa;            /*W-force drag coefficient due to aoa       */
        float   CLo;            /*reference lift coefficient          */
        float   CLdalpha;       /*delta alpha */
        float   CLq;            /*lift due to pitch moment*/
        float   CLa;            /*lift due to angle of attack*/
        float   CLde;           /*lift due to elevator deflection*/
        float   CMo;            /*pitch moment coefficient at 0 alpha  */
        float   CMq;            /*pitch moment coefficient due to pitch rate  */
        float   CMa;            /*pitch moment coefficient due to aoa         */
        float   CMda;           /*pitch moment coefficient due to delta aoa  */
        float   CMde;           /*pitch moment coefficient from elevator motion*/
        float   CYb;            /*Y-force coefficent due to side slip angle  */
        float   CYdr;           /*Y-force coefficent due to change in yaw rate */
        float   CLr;            /*roll moment coefficient due to yaw rate    */
        float   CLp;            /*roll moment coefficient due to roll rate   */
        float   CLda;           /*roll moment coefficient due to change in ail */
        float   CLb;            /*roll moment coefficient due to side slip   */
        float   CLdr;           /*roll moment coefficient due to change in yaw */
        float   CNda;           /*aoa change effect on yaw moment       */
        float   CNb;            /*sideslip change effect on yaw         */
        float   CNp;            /*roll rate effect on yaw moment        */
        float   CNr;            /*yaw rate effect on yaw moment         */
        float   CNdr;           /*yaw acceleration effect on yaw moment      */
        float   def_rud;        /*rudder deflection +/- in radians          */
        float   def_ail;        /*aileron deflection +/-  in radians    */
        float   def_elev;       /*elevator deflection +/- in radians          */
        } FLIGHTSPECS;
```

Figure C.3: Aircraft Specification Data Structure

C. INITIALIZATION OF THE AIRCRAFT DATA STRUCTURE

To give the system user a convenient method of assigning aircraft specification data to each aircraft in NPSNET, a data input file (aero.dat), together with "read_acftdat.c", is used. One simply enters the data for a particular type of aircraft in the data input file and

specifies, in the type aircraft field, a particular vehicle type. The vehicle types have been previously defined in NPSNET for distinguishing between types of vehicles. If an aircraft in NPSNET has a type number that does not match a type number entered in the data file then a default type is assigned by way of the "read_acftdat.c" program. This default type provides the "easiest" aircraft to pilot. An unlimited number of types can be entered through this data file. Users should insure that the number of these records are specified at the top of the file (Figure A3.4).

```
H                /label
1                /# of aircraft types

t                /record type/
30               /match to type of aircraft/
1                /jet or prop jet:1 prop:0/
27.5             /b/
260.0            /S/
10.8             /c/
546.0            /m/
8090.0           /Ix/
25900.0          /Iy/
29200.0          /Iz/
1300.0           /Ixz/
0000.0           /max thrust/
8000.0           /mil thrust/
 0.03   0.3                          /CDo /CDa
 0.28   3.45   0.0    0.72   0.36    /CLo /CLa /CLq /CLda /CLde
 0.0    -3.6   -0.38  -1.1   -0.5    /CMo /CMq /CMa /CMda /CMde
-0.98   0.17                         /CYb /CYdr
-0.12   -0.26  0.14   0.08   -0.105  /CLb /CLp /CLr /CLda /CLdr
 0.25   0.022  -0.35  0.06   0.032   /CNb /CNp /CNr /CNda /CNdr
 0.2618 -0.5236 0.5236               /deflection of rud, ail + stab
```

Figure C.4: Aircraft Data Record

"Read_acftdat.c" contains the code for linking the aircraft structures together. It takes the entire vehicle array, determines which vehicles are aircraft, matches the type records in the data file with the specified vehicle types, initializes the two data structures (above), and places an updated vehicle structure back into NPSNET.

D. COCKPIT DISPLAY

In order to have an appropriate interface between the user and the aircraft simulation, a rudimentary cockpit was devised and inserted in the code at the end of the procedure "graphicsproc()" found in jeep.c.

The projection of the cockpit is ortho2 and the following code was added:

```
prefposition(0,XMAXSCREEN,0,YMAXSCREEN
winset(jeepwin);
acft_cockpit(vehpos);
```

For control of the radar sweep the following global variables were added:

```
float   deltatime;      /*frames per second*/
float   radar_pos;      /*keeps track of radar look angle*/
float   rdr_posx;       /*radar sweep x axis position*/
int     reverse_dir;    /*radar scan direction*/
```

The following definitions were placed in the "aero.h" file:

```
LEFTSCOPE 48.5
RIGHTSCOPE 59.0
TOPSCOPE  14.0
BOTTOMSCOPE 3.5
SCANRATE  72.0
SCANWIDTH 120.0
SCOPEWIDTH  (RIGHTSCOPE-LEFTSCOPE)
HALFSCANWIDTH (0.5*SCANWIDTH)
SCOPERATE  (SCANWIDTH/SCOPEWIDTH)
CENTERSCOPE  (LEFTSCOPE+(SCOPEWIDTH/2.0))
COMPASSRADIUS 5.0
TEN  (COMPASSRADIUS-0.5)
THIRTY  (COMPASSRADIUS-1.0)
```

E. INPUT DEVICE MODIFICATION FOR AIRCRAFT CONTOL

1. Spaceball

Control input for an aircraft is limited to movement of the Spaceball along the forward-aft axis and the left-right axis. Therefore, the method in which the Spaceball data is read was modified. To accomplish this, the procedure, newsbvals() in jeep.c, was separated by an 'if' statement into two cases: (1) an aircraft and (2) not an aircraft. From the Spaceball data array, values [0] and [2] are converted into aileron and elevator position data. The range of values is limited to -1.0 and 1.0.

2. Keyboard

The keyboard has been modified to add rudder, elevator trim and thrust input. The ">" and "<" keys control the rudder input. The rudder can be moved left and right in .05 radian increments, and maintains a value between -1.0 and 1.0.

Elevator trim is controlled by the 'e'/'E' and 'r'/'R' keys. 'e'/'E' trims the elevator surface up and 'r'/'R' trims the elevator surface down. Thrust is controlled by the 's'/'S','a'/'A',and 'd'/'D' keys, and has a value between 0.0 to 1.0. The 's'/'S' key moves the throttle higher in 0.05 unit increments, the 'd'/'D' key moves the throttle higher in 0.3 unit increments, and the 'a'/'A' key moves the throttle down in 0.05 unit increments.

F. INTEGRATION OF THE AERODYNAMIC MODEL

The aerodynamic model used can be found in Appendix A and is incorporated as a separate compilation file "aero.c". Within "jeep.c" a test is made as to whether the vehicle being controlled is a ground or air vehicle. If it is an air vehicle, program control moves to procedure movetheaircraft() found in "jeepmot.c", where the procedure call, aero_model(), is made. The aerodynamic model functions exactly as described in the prototype simulator. After returning from procedure aero_model(), position updates are calculated and all the variables used by NPSNET are assigned new values.

One aspect of NPSNET is the ability to switch from one vehicle to another through the planar map display. Since the vehicle starts out with an initial heading and airspeed, it is important to "remember" this information when assuming control of the vehicle. This is accomplished by initializing the aircraft state structure (Figure C.2) with airspeed and attitude data within the procedure switchveh() in file "dogsncats.c". It is important to remember at this point that the aerodynamic model will convert all zero airspeed values to 10 ft/sec. If the desire is to have an immediately flyable plane at the vehicle switch, then the airspeed should be initialized to at least 100 ft/sec.

G. INTEGRATION OF ORIENTATION MODEL

The orientation model can be found in file aero.c in two separate places. The first place is at the end of procedure aero_model(), where it is used to determine pitch, roll and yaw data for the driven aircraft. The second place is in the same file but in a separate procedure, rotate_translate_acft(). The procedure rotate_translate_acft() can be used in future

NPSNET improvements when a quaternion rotation model is desired for updating the orientation and positioning of all non-driven vehicles in NPSNET (Figure C.5).

```
rotate_translate_acft(ACFTPTR P, float dtime) /*framerate*/
{
 Matrix A_MATRIX;
 float vel_wor[3];

 /******incrementally rotate quaternion*********************/
 increment_quat(P->ang_vel[0], P->ang_vel[1], P->ang_vel[2],
 P->quat,dtime);

 /*****update rotation matrix from quaternion***************/
 rotation_matrix_from_quat(P->quat, P->h_matrix);

 /*****extract euler angles from matrix*********************/
 euler_angles_from_matrix(&P->sroll,&P->croll,
                          &P->sptch,&P->cptch,
                          &P->shdg, &P->chdg, P->h_matrix);
 P->euler_angles[1] = (fasin((P->sptch)));
 P->euler_angles[0] = (fasin((P->sroll)));
 if (P->croll < 0.0)
 if (P->sroll < 0.0) P->euler_angles[0] = (-PI - P->euler_angles[0]);
 else P->euler_angles[0] = (PI - P->euler_angles[0]);

 P->euler_angles[2] = (fasin((P->shdg)));
 if (P->shdg < 0.0)
 {
 if (P->chdg < 0.0) P->euler_angles[2] = PI - P->euler_angles[2];
 else P->euler_angles[2] = TWOPI + P->euler_angles[2];
 }
 else
 if (P->chdg < 0.0) P->euler_angles[2] = PI - P->euler_angles[2];

 /********calculate new world velocity in sgi coords**********/
 transpose_matrix(P->h_matrix,A_MATRIX);
 post_mult_matrix_by_vector(A_MATRIX,P->lin_vel,vel_wor);
 P->vel_world[0] = vel_wor[1] * FT_TO_METERS;
 P->vel_world[1] = -vel_wor[2] * FT_TO_METERS;
 P->vel_world[2] = -vel_wor[0] * FT_TO_METERS;

 /****calculate new world position is sgi coords*****************/
 P->h_matrix[3][0] += P->vel_world[0] * dtime;
 P->h_matrix[3][1] += P->vel_world[1] * dtime;
 P->h_matrix[3][2] += P->vel_world[2] * dtime;
 P->h_matrix[3][3] = 1.0;

 /********calculate world lookat point*****************/
 P->lookatpt[0] = P->pos[0] + (cosptch * coshdg);
 P->lookatpt[1] = P->pos[1] + sinptch;
 P->lookatpt[2] = P->pos[2] + (cosptch * sinhdg);
)
```

Figure C.5: Procedure rotate_translate_acft()

APPENDIX D

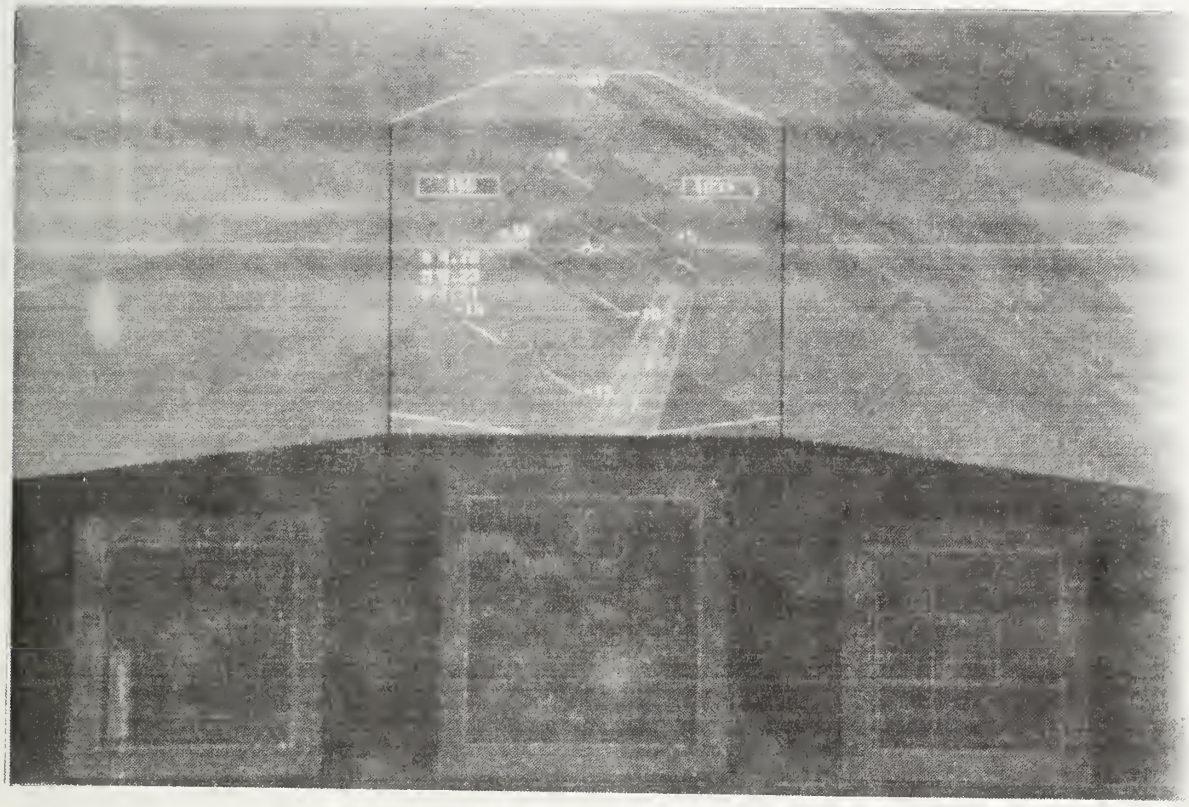

Photo 1: Passing over the Airport

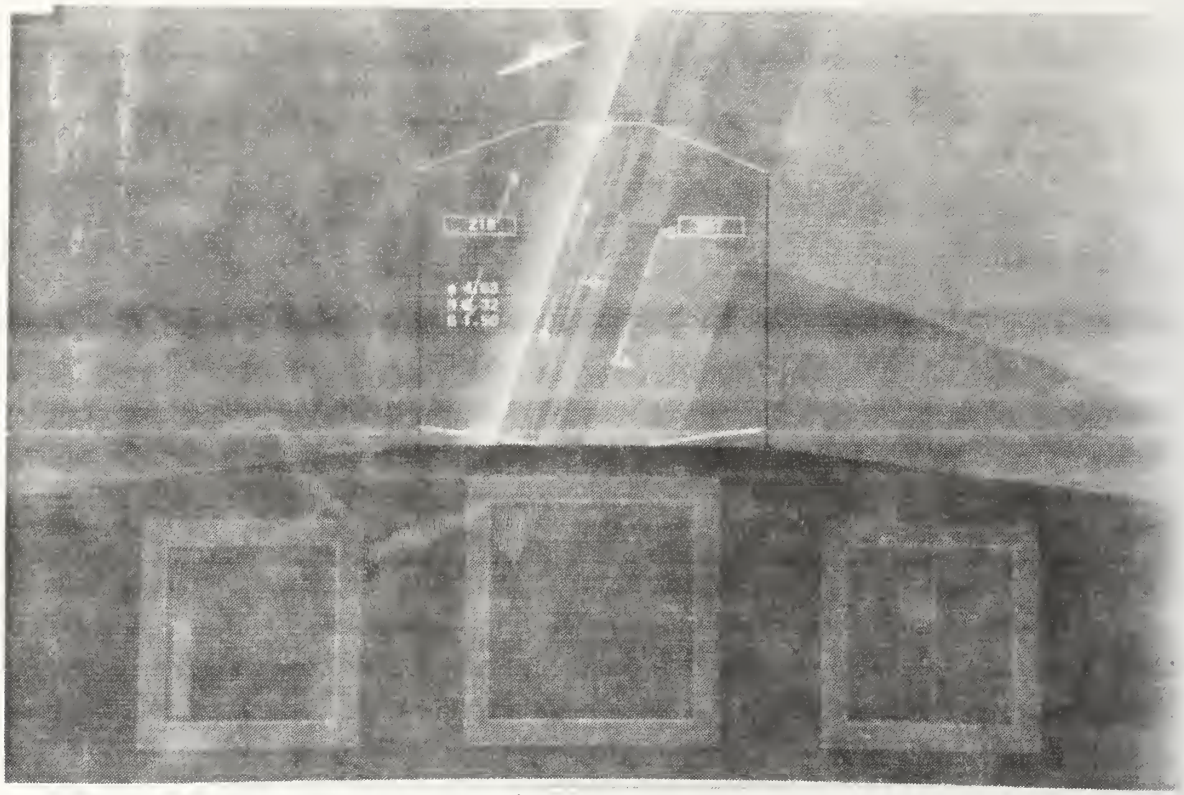

Photo 2: Approaching an Aircraft Turning Through North

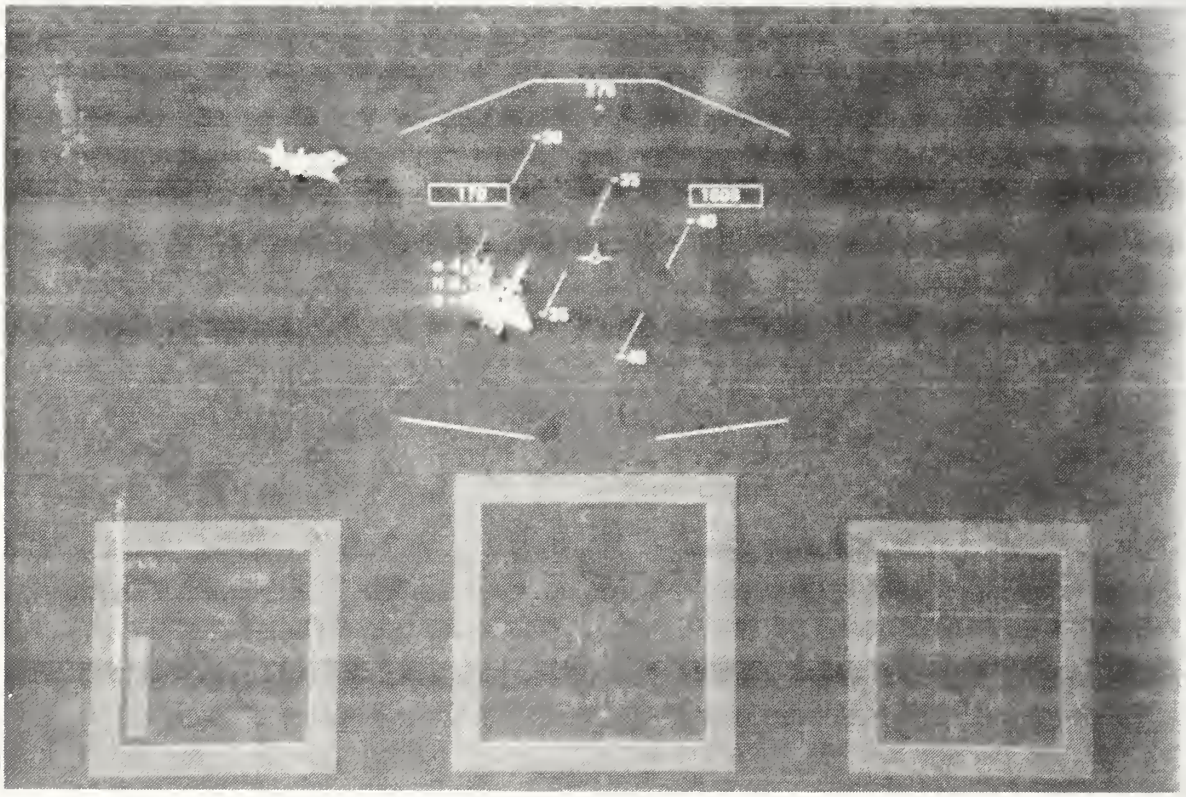

Photo 3: Target of Opportunity

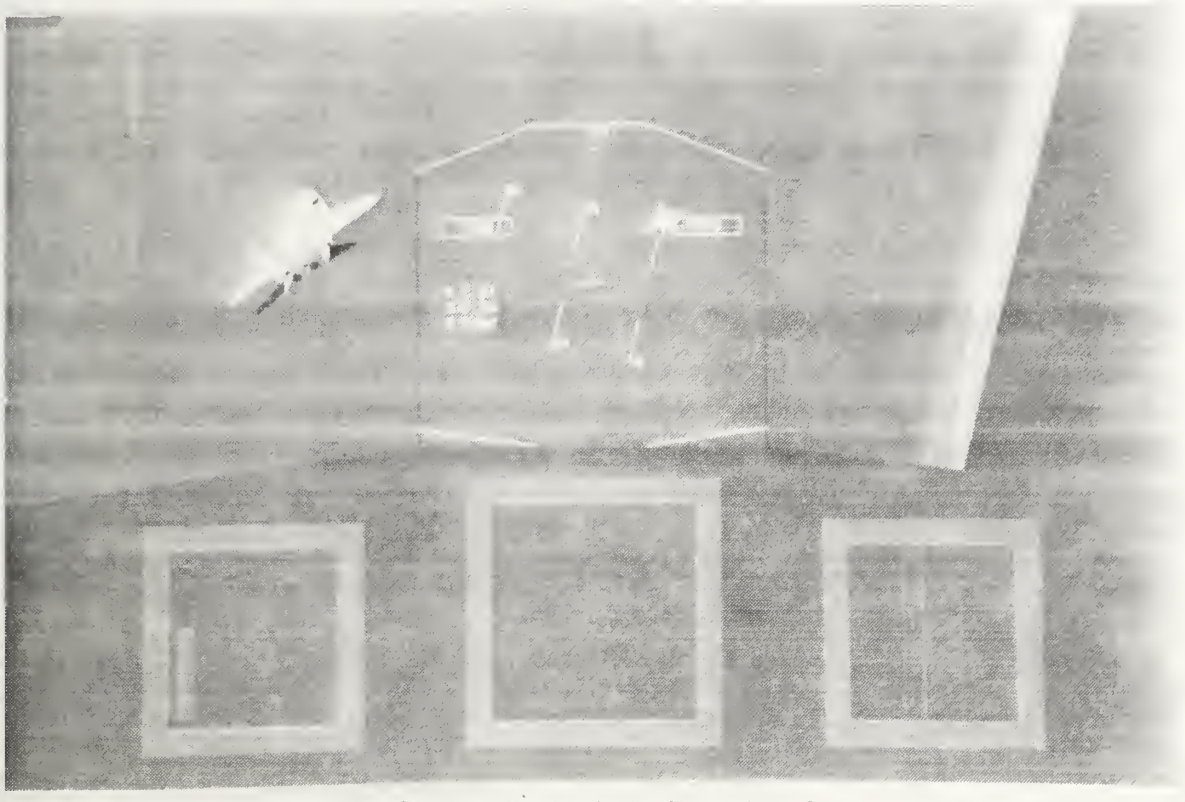

Photo 4 Closing in On Two Aircraft

LIST OF REFERENCES

Anderson, John D., *Introduction to Flight,* McGraw-Hill Publishing Company, New York, NY, 1989.

Burchfiel, J., "The Advantages of Using Quaternions Instead of Euler Angles for Representing Orientation", White Paper ASD-91-001, Third Workshop on Standard for the Interoperability of Defense Simulations, Orlando FL, August,1990.

Goldiez, Brian and Lin, Kuo-Chi, "The Orientation Representation in the Draft Military Standard for Distributed Interactive Simulation", *AAI,* University of Central Florida, Orlando, FL, 1991.

Goldstein, Herbert, *Classical Mechanics*, Second Ed., Addison-Wesley Publishing Co., Inc., Reading, MA, 1980.

Mitchell, E.E., Rodgers, A.E., "Quaternion parameters in the Simulation of a Spinning Rigid Body, *Simulation*, 18, No 6, 1965.

Nelson, Robert C., *Flight Stability and Automatic Control*, McGraw-Hill Book Company, New York, NY, 1989.

Paul, Richard P., *Robot Manipulators: Mathematics, Programming, and Control,* The MIT Press, The Massachusetts Institute of Technology, 1981.

Press, W.H.,Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., *Numerical Recipes in C: the Art of Scientific Computing,* Cambridge University Press, 1990.

Rolfe, J. M., Staples, K. J., *Flight Simulation*, Cambridge University Press, 32 East 57th Street, New York, NY, 10022, 1986.

Roskam, J., *Airplane Flight Dynamics and Automatic Flight Controls*, Roskam Aviation and Engineering Corporation, 1979.

Shoemake,K., Animating Rotation with Quaternion Curves, *Computer Graphics*,19(3), 245-254, SIGGRAPH Conference Proceedings, July 1985.

Thorpe, Jack A., "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting," *Proceedings of the Ninth Interservice Industry Training Systems Conference*, November 1987.

UCF/IST, "Military Standard (DRAFT) for Protocol Data Units for Distributed Interactive Simulation". University of Central Florida Institute for Simulation and Training. June 1990.

Zyda, Michael J., Pratt, David R., Monahan, James G. and Wilson, Kalin P., "NPSNET: Constructing a 3D Virtual World", Proceedings of the 1992 Symposium on Interactive 3D Graphics. March, 1992.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 221304-6145	2
2.	Commandant of the Marine Corps Code TE06 Headquarters, U.S. Marine Corps Washington, D.C. 20380-001	1
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Michael J. Zyda Naval Postgraduate School Code CS, Department of Computer Science Monterey, CA 93943-5100	5
4.	David Pratt Naval Postgraduate School Code CS, Department of Computer Science Monterey, CA 93943-5100	2
5.	Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
6.	Major Joseph M. Cooke 1292 Spruance Monterey, CA 93940	1

Thesis
C74765 Cooke
c.1 NPSNET : flight simulation dynamic modeling using quaternions.